OPTIMAL BANDWIDTH ALLOCATION AND CONTROL FOR NETWORKED CONTROL SYSTEMS WITH DISTURBANCE AND NOISE

A Dissertation

by

KUKTAE KIM

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Won-jong Kim
Committee Members,	Reza Langari
	Alexander G. Parlos
	Sergiy Butenko
Head of Department,	Andreas Polycarpou

August 2019

Major Subject: Mechanical Engineering

ABSTRACT

A networked control system (NCS) is an interconnected control system in which sensors, actuators, and controllers communicate with each other through a shared network. Although NCSs are beneficial thanks to easy maintenance, architectural flexibility, decreased wiring weights, and tele-operating possibilities, NCSs also have some challenges such as disturbance, noise, bandwidth limitation, delay, and packet dropout. The popularization of smartphones and the drastically increasing number of internet of things (IoT) devices require not only a high-speed internet such as 5G, but also a wise strategy for optimal bandwidth allocation. In this dissertation, optimal bandwidth allocations for NCSs with disturbance and noise are proposed based on performance index function (PIF), artificial neural network (ANN), and Q-learning algorithms. A ball magnetic-levitation (maglev) system, four DC motor speed-control systems, and a wireless autonomous robotic wheelchair are implemented as test beds.

The relationship between system performance, sampling frequency, and the standard deviation of white Gaussian disturbance are approximated using a 6^{th}-degree polynomial. The PIF and ANN methods can estimate the standard deviation of disturbance when current a sampling frequency and an error variance are provided. Dynamic bandwidth allocation using PIF, ANN, and Q-learning is proposed and verified by experimental results for a single-server and single-client DC motor system. The proposed methods show integral absolute errors (IAE) of 166 615, 16 773, and 16 945 and bandwidth utilizations (BU) of each method are 13.15%, 13.38%, and 13.98%,

respectively, after 15 000 iterations, when various standard deviations of disturbance are injected. These results present a better performance and a reasonable average BU compared to fixed sampling frequencies. When information of the estimated standard deviation of disturbance, BU margin of safety, weight of each system, and total time delay is given, the optimal sampling frequency for a multi-server and multi-client system can be determined based on the PIF, ANN, and Q-learning, respectively. They are validated by experiments in two cases. The first case is conducted with a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, and various weights for four DC motor systems. The second has the conditions of a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, various weights for four DC motor systems with a maglev and a wheelchair robot system, of which BUs are 44% and 1% respectively. Experimental results prove that all three methods can be used to find the optimal sampling frequencies for each system when an NCS has limited bandwidth as well as sufficient bandwidth.

To the Almighty God,

my parents (Jongmok Kim, Hyesuk Cha),

and beloved wife (Jungjin Kim)

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Dr. Won-jong Kim for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. I could not have imagined having a better advisor and finishing my Ph.D study without him.

Besides my advisor, I would like to thank the rest of my thesis committee: Drs. Reza Langari, Alexander G. Parlos, and Sergiy Butenko for their insightful comments and encouragement.

I would like to thank my fellow labmates, Youngshin Kwon, Jiawei Dong, Vu Nguyen, Abdullah Algethami, Jinfa Chen, Ivan Cortes, and James McCabe for all the fun we have had. Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Last but not the least, I would like to thank my mother, Hyesuk Cha and father, Jongmok Kim for supporting me throughout my life in general and to my wife, Jungjin Kim for her patience and love.

CONTRIBUTORS AND FUNDING SOURCES

This work was supervised by a dissertation committee consisting of Professors Won-jong Kim, Reza Langari, and Alexander G. Parlos of the Department of Mechanical Engineering and Professor Sergiy Butenko of the Department of Industrial and Systems Engineering.

All work for the dissertation was completed independently by the student.

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

NOMENCLATURE

3-D	Three-Dimensional
ANN	Artificial Neural Network
BP	Backpropagation
BU	Bandwidth Utilization
CAN	Controller Area Network
Comedi	Control and Measurement Device Interface
CPU	Central Processing Unit
DAQ	Data Acquisition
DC	Direct Current
DNA	Deoxyribonucleic Acid
IAE	Integral Absolute Error
ICCS	Integrated Communication and Control Systems
IoT	Internet of Things
ITAE	Integral Time-weighted Absolute Error
KKT	Karush-Kuhn-Tucker
LAN	Local Area Network
LMI	Linear Matrix Inequality
Maglev	Magnetic levitation
MEMS	Micro Electro Mechanical Systems
MSE	Mean Squared Error

NASA	National Aeronautics and Space Administration
NCS	Networked Control System
NI	National Instruments
OS	Operating System
PC	Personal Computer
PI	Proportional Integral
PIF	Performance Index Function
PWM	Pulse-Width Modulation
rps	revolutions per second
RTAI	Real-Time Application Interface
RTOS	Real-Time Operating Systems
SQP	Sequential Quadratic Programing
UDP	User Datagram Protocol
WGN	White Gaussian Noise
WLAN	Wireless Local Area Network

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION AND LITERATURE REVIEW

The Internet of Things (IoT) is the connected network via the Internet of physical objects or things embedded in electronics, software, and sensors. It can transfer data to the manufacturer, operator, and other connected devices through the Internet without requiring human-to-human or human-to-computer interaction. IoT evolved from the convergence of wireless technologies, micro-electromechanical systems (MEMS), and the Internet. Recently, it became one of the most studied topics [1]–[6]. Since it was first invented, the Internet has become one of the most significant creations and plays an important role in many areas such as education, communication, business, science, and government. Cisco Systems, Inc., expects that the implementation of the IoT will connect 50 billion smart objects to the Internet by 2020 [1]. Thus, bandwidth allocation of networked control systems (NCSs) and managing data communication is the key of the IoT for connected devices.

1.1 Networked Control Systems

NCSs are defined as spatially distributed systems in which sensors, actuators, and controllers communicate with each other through a shared network. With the development of high-speed Internet and wireless connections, NCSs gained significant attention recently due to significant advantages such as easy maintenance, architectural flexibility, decreased wiring costs, tele-operating possibilities, etc. The design of an NCS requires knowledge of not only control and hardware engineering but also computer science, software engineering, and real-time operating-systems (RTOS) due to the introduction of

the networks [7]. In the NCSs, these two fields are correlated, and their separation will lead to the degradation of system performances.

An NCS can consist of multiple servers or clients and each server or client can be connected to hardware. Figure 1-1 illustrates general examples of an NCS from several possible operation scenarios between servers and clients such as a single server and a single client, a single server and multiple clients, multiple servers and a single client, and multiple servers and multiple clients. When multiple plants or clients are connected in the same network and communicate, the NCS is needed to allocate limited resources such as network bandwidth, central processing unit (CPU) speed, and battery power for mobile devices to maintain stability and allowable performance.

1.2 Literature Review

1.2.1 History of NCS

The analog fly-by-wire flight control system for the Avro Vulcan in the 1950s can be called the first form of analog NCS because the system was designed to eliminate the complexity, fragility, and weight of the hydro-mechanical flight control systems by using an electrical circuit. As digital computers and microprocessors acquired more power in computing, they accelerated the capability of NCSs in control systems [8]. The first digital fly-by-wire aircraft, a modified National Aeronautics and Space Administration (NASA) F-8C Crusader, was developed in 1972. The first published research papers of the control over networks were written by Halevi and Ray in 1988 [9], [10]. They addressed time-varying and possibly stochastic delays in Integrated Communication and Control Systems

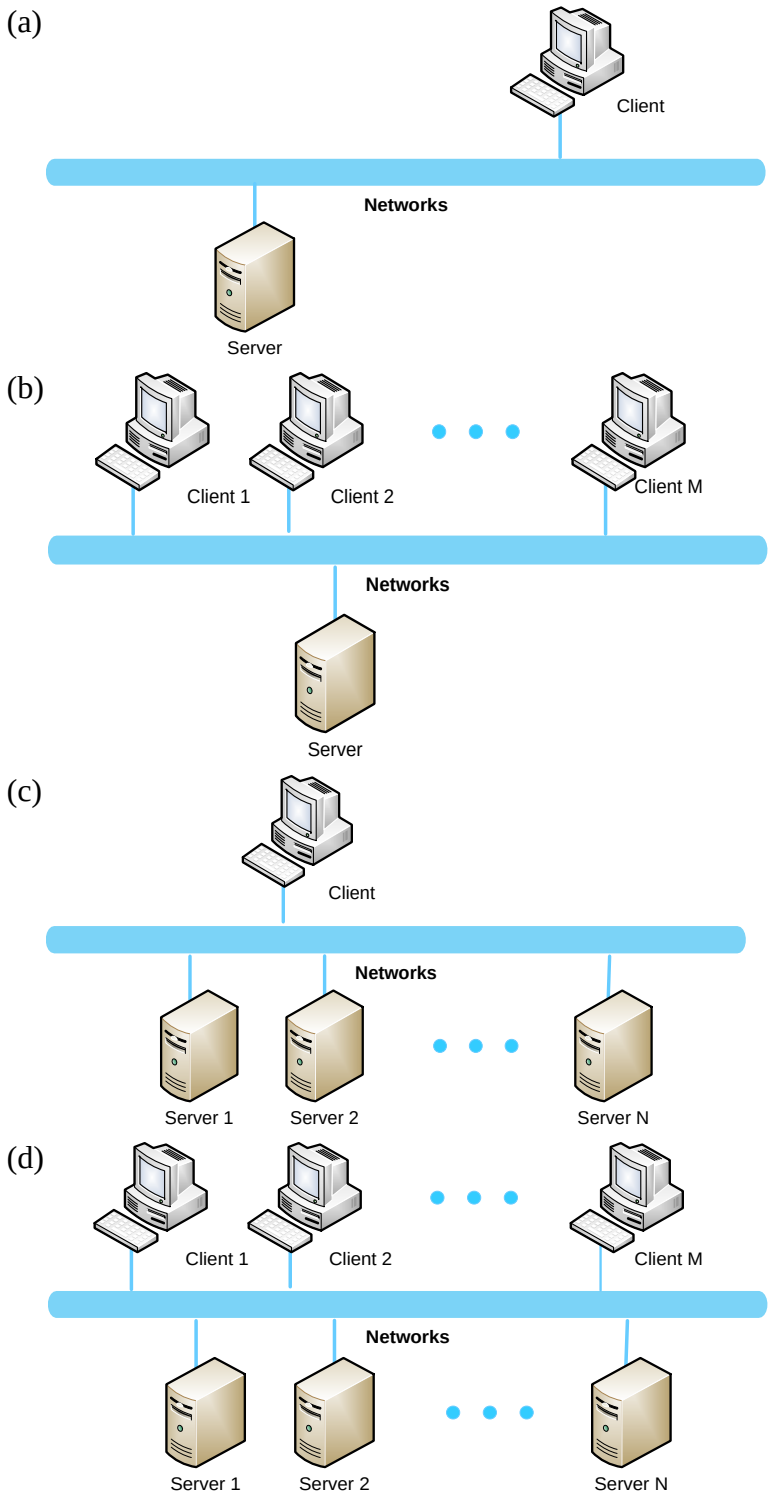

Figure 1-1. A general architecture of an NCS: (a) a single-server and a single-client NCS (b) a single-server and multi-client NCS (c) a multi-server and a single client NCS (d) a multi-server and multi-client NCS

(ICCS) and design considerations for ICCS. After that, NCSs have been actively studied and used in many fields. Zhang *et al*. studied network-induced constraints of the NCSs including time delays, packet losses, resource competition, data quantization, etc. [11]. A state-estimation problem involving several finite communication capacity constraints was discussed by Wong and Brockett [12], [13].

1.2.2 Time delays and packet droupouts

In the NCSs, network induced delays and packet dropouts are inevitable. Thus, they are one of the most popular research topics in the NCS field. Yu *et al*. proposed to model NCSs with arbitrary but finite data packet dropouts and network delays using a switching system approach [14]. An linear matrix inequality (LMI) approach to networked control systems with data packet dropouts and transmission delays are presented by Yu *et al*. [15]. They proposed a Lyapunov-Razumikhin-based method for the continuous-time case and a Lyapunov-Krasovskii-based method for the discrete-time case. Zhang *et al*. presented a new switched linear system model to describe NCSs with a network delay and packet dropout at the same time [16]. Xiong *et al*. presented the stabilization of linear systems over networks with bounded packet loss [17]. The arbitrary packet-loss process and the Markovian packet-loss process were considered, and the corresponding stabilizing controller design techniques were given based on the Lyapunov approach. Sahebsara *et al*. proposed optimal H_2 filtering with random sensor delays, multiple packet dropouts, and uncertain observations [18]. A fuzzy model-based robust estimator was studied by Zhang *et al*. where the plant is a nonlinear systems with external disturbances [19]. Wang

et al. considered the robust H_∞ control problem for NCSs with random communication packet losses [20].

1.2.3 Bandwidth allocation and scheduling

Not only software design but also implementation conditions such bandwidth, sampling frequency, and performance of devices should be integrated to design an NCS. Velasco *et al*. proposed a dynamic control approach managing the bandwidth of the NCS based on the dynamics of controlled processes [21]. Optimal tracking performance issues with limited bandwidth and additive colored white Gaussian noise was researched by Guan *et al*. [22]. Castane *et al*. presented a resource management strategy for the NCS that maximized control performance within available resources [23]. A dynamic bandwidth allocation algorithm for a video based network system was proposed to raise the bandwidth utilization (BU) of an NCS [24]. Belzarena *et al*. presented the network bandwidth allocation with time reservations [25]. Branicky *et al*. recommended a co-design approach to treat communication protocols and interacting controlled plants as a coupled system [26]. An approach was introduced to implement the dynamic scheduling policy for the controller area network (CAN) bus with the system performance guaranteed by Weiss *et al*. [27]. Xu *et al*. formulated a bandwidth optimization and scheduling algorithm for an NCS using a non-cooperative game model and designed the utility function of susbsystems [28].

1.2.4 Performance index function

Unlike continuous or digital control systems, a higher sampling frequency does not always guarantee better performance in an NCS. Lian *et al*. studied performance index functions (PIFs) related to the sampling frequency and performance [29], [30]. They presented several key components of the time delay analysis, and the analysis of NCS parameters was used to determine an acceptable working range of the sampling frequency in an NCS. These performance characteristics are crucial guidelines for choosing the network and control parameters in the design of an NCS because a network has a limited communication bandwidth, and a higher sampling frequency to operate devices consumes more energy than a lower sampling frequency. Operating the hardware system at a higher frequency also increases costs. Dong and Kim discussed an optimal bandwidth allocation for NCSs with nonlinear approximation techniques [31]. The optimal sampling frequencies are obtained by solving the exponential approximation of PIFs with the Karush-Kuhn-Tucker (KKT) method. Because of continuing popularity for communication with IoT devices despite of weakness in wireless networks, disturbance and noise should be considred when designing an NCS. Kim and Kim studied PIFs in the NCSs with disturbance and noise [32].

1.2.5 Artificial neural network

ANN are used in various areas such as modelling and identification and control of nonlinear systems [33]. Yi *et al*. used a backpropagation (BP) feedforward neural network to predict the time delay induced in NCSs [34].

1.2.6 Machine learning

Although machine learning methods were defined in 1959 by Samuel [35], they were not actively studied until the middle of the 1990s as one of the AI fileds. The recent five-game Go match, AlphaGo versus Lee Sedol, ignited global attention about machine learning [36]. Machine learning focuses on prediction making using complex models and algorithms, which is devised from experience. It is widely used in areas such as robotics, financial services, health care, oil and gas, marketing and sales, classifying deoxyribonucleic acid (DNA) sequences, computer vision, and game playing. Dorigo and Schnepf presented genetics-based machine learning and behavior-based robotics [37]. Nagumo and Noda suggested a method for system identification using a learning identification based on an error correcting procedure [38]. Panigrahi and Phandi studied an optimal feature selection for classification of power quality disturbances using a wavelet packet based fuzzy k-nearest neighbor algorithm [39]. A command-based iterative learning control algorithm to reduce the effects of friction, disturbance, and noise was studied by Tsai *et al*. [40]. In the NCS field, machine learning methods has not been widely used. Most of the work focuses on detecting the anomaly of the network [41], [42]. Jang *et al*. studied the optimization of the uplink period using machine learning for the future IoT network using a Naïve Bays classifier [43]. A self-tuning fuzzy controller for an NCS was proposed by Tian et al. [44].

1.3 Research Issues

1.3.1 Bandwidth limitation

Any communication network can only carry a finite amount of information per time unit. This limitation poses significant constraints on the operation of NCSs. Since general NCSs have multiple sensors and actuators to control the systems, optimal and efficient utilization of available bandwidth in a network should be considered [45]. The Shannon-Hartley theorem expresses the maximum bit rate that a communication channel can transmit with the presence of noise. In most digital networks, data are transmitted in packets and consume the same amount of network resources whether sending a single bit or hundreds of bits. Significant research has also been done to determine the minimum bit rate necessary to stabilize linear systems through feedback [12], [46]–[49] and nonlinear systems [50], [51] over a finite capacity channel.

1.3.2 Time delay and packet dropout

Unlike traditional continuous-time control, to transmit a continuous-time signal over a network in NCSs, the signal is sampled at first, encoded in a discrete format, transmitted over the network, and decoded on the receiver side as shown in Fig. 1-2. The entire delay between sampling and final decoding at the receiver can be highly stochastic because both the network access delays and the transmission delays depend on not only variable network conditions such as congestion and channel quality but also the computation time required for physical coding. The time delays can degrade the performance of an NCS and even destabilize the system when the severe delay induces

data-packet loss during data transmission [52]–[55]. Packet dropouts can originate from transmission errors in physical networks, which is much more frequent in wireless than in wired networks, or from buffer overflows due to data congestion.

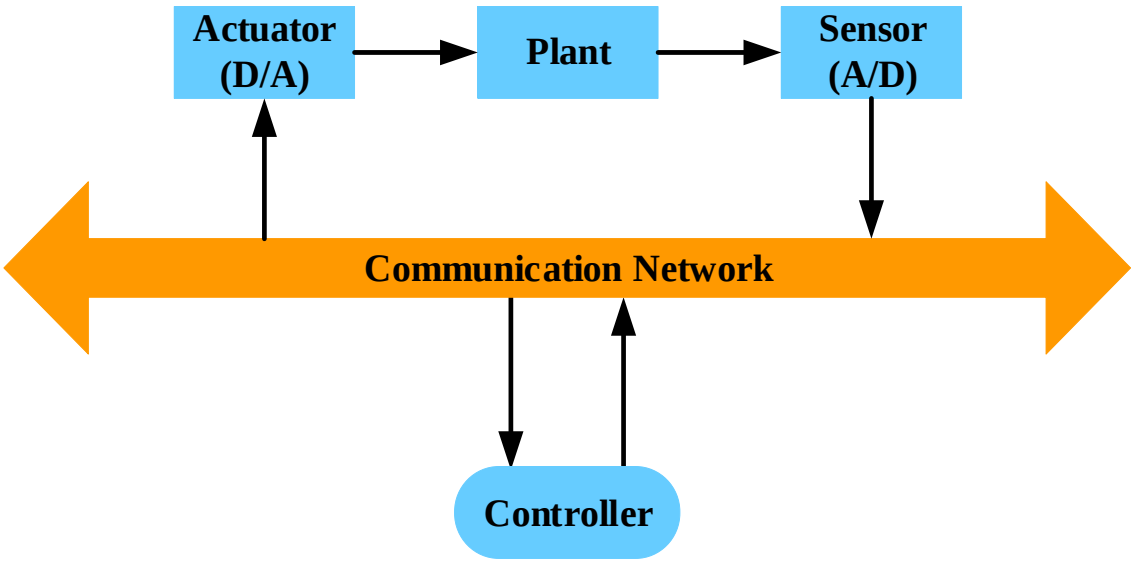

Figure 1-2. Block diagram of an NCS

1.3.3 Disturbances or noises in NCSs

Disturbances or noises are any undesirable perturbation or interference that negatively affects the output of the control system and results in increasing the system error. They are commonly found in control systems and are usually used interchangeably with a following additional explanation. In this dissertation, for clarification, they will be distinguished as an input disturbance and a measurement noise based on their entry points in the closed control loop.

In many cases, an input disturbance is in a low-frequency range or a constant value and has a certain amplitude level at the input of a control system. A measurement noise is an unwanted signal, which is generally small in magnitude and high in frequency, riding on top of the desired signal. It usually happens to the measurement instruments such as a sensor. They can be generated from within the system itself or an outside source.

There are various types of noises or disturbances that have been studied such as white noise, white Gaussian noise, pink or flicker noise, brown or red noise, blue noise, violet noise, grey noise, Poisson noise, shot noise, and burst noise [56], [57]. This unpredictability makes the design of an NCS quite challenging.

1.3.3.1 White noise

White noise is a random signal in which the frequency and power spectrum is constant and independent of frequency. Its name comes from a similarity to white light, which has uniform emissions at all frequencies. When plotted in a frequency domain, white noise is a horizontal line with a constant value. If it has a normal distribution in the time domain with zero mean, it is called Gaussian white noise.

1.3.3.2 Pink noise

Pink noise is a signal with a frequency spectrum such that the power spectral density is inversely proportional to the frequency (f) of the signal. It carries equal energy per decade. This means that the amplitude decreases logarithmically with the frequency. Flicker noise also displays a $1/f$ characteristic and rolls off at a 20 dB/decade.

1.3.3.3 Red noise or Brown noise

Brown noise, also known as red noise, is a kind of signal noise produced by Brownian motion. The term Brown noise refers to Robert Brown, the discoverer of Brownian motion, it was not named for the color. It has a –40 dB/decade frequency response and a frequency spectrum of $1/f^2$.

1.3.3.4 Other noises

The power density of blue noise increases at 20 dB/decade with a frequency spectrum of f and the power density of violet noise increases at 40 dB/decade with a frequency spectrum of f^2. Grey noise's power density decreases at first and increases after a certain frequency. Its behavior looks like combination of pink noise and blue noise. Poisson noise or Shot noise is a type of electronic noise which can be modeled by a Poisson process. It is caused by a random fluctuation in the motion of charge carriers in a conductor. Burst noise or popcorn noise in signal processing consists of sudden step-like transitions between two or more discrete signal levels at random and unpredictable times. Each shift in offset lasts from several milliseconds to seconds. It sounds like popcorn popping as its name indicates.

1.4 Contribution

This dissertation focuses on optimal bandwidth allocation and control for NCSs with disturbance and noise. The NCS in this research consists of a ball maglev system, a DC motor speed control system that contains four DC motors, and an autonomous wireless wheelchair robot as test beds to verify proposed control methodologies and algorithms,

and optimal bandwidth allocation. Each client has a unique identification number within its data packets to distinguish one from the other. We employ an Ethernet based local area network (LAN) as a communication network. User datagram protocol (UDP) is applied as the communication protocol in the NCS. Various control issues involved in the NCS and importance of optimal bandwidth allocation are studied.

Major accomplishments of this research are as follows: (1) Optimal bandwidth allocation and scheduling of the NCS to guarantee the system performance of each client in the NCS is presented based on PIF. (2) Optimal bandwidth allocation and scheduling of the NCS to guarantee the system performance of each client in the NCS is presented based on ANN. (3) Optimal bandwidth allocation and scheduling of the NCS to guarantee the system performance of each client in the NCS is presented based on Q-learning method.

Each bandwidth allocation algorithm will distribute the network bandwidth to each client to achieve the maximum system performance.

1.5 Dissertation Organization

This dissertation contains seven sections. Section 1 briefly describes the concept of NCSs and their general architectures. Current research issues are also presented. This section provides a literature review of the research topic.

Section 2 presents research methodologies to solve the problems identified in section 1.

Section 3 explains the experimental setup. Details of the ball maglev system, wheelchair robot system, DC motor system, and their initial tests are provided.

Section 4 discusses the PIF for the NCS with disturbances and noises. Polynomial and exponential approximation methods are suggested to estimate the standard deviations of disturbances and noises. An optimal bandwidth allocation method for multi-server and multi-client NCSs with disturbances and noises using PIFs is also proposed in this section.

Section 5 examines the ANN method to estimate disturbances and noises in the NCSs and to determine optimal sampling frequencies to minimize error variances and BU for multi-server and multi-client NCSs.

Q-learning algorithm is introduced for optimal bandwidth allocation for multi-server and multi-client NCSs with disturbances and noises in section 6.

Section 7 concludes this dissertation summarizing its achievements and provides suggestions for possible future work.

2. RESEARCH METHODOLOGIES

2.1 Research Objectives and Assumptions

The main objectives of this research can be summarized as follows:

1) Suggest an optimal network bandwidth allocation method to distribute bandwidth to each client in an NCS when disturbance and noise are considered as the parameters of the PIF. The proposed method will show a dynamic and flexible sampling frequency adjustment to meet the system requirements in real time and maintain the maximum system performance if possible.

2) Propose an ANN method to find an optimal network bandwidth allocation in an NCS with disturbances and noises in real time. Unlike the PIF, the ANN does not need to calculate hundreds of iterations to find optimal sampling frequencies and will reduce the calculation time.

3) Introduce a Q-learning method to an NCS with disturbances and noises to determine an optimal network bandwidth allocation. The Q-learning will be advantageous in that it does not need experimental data to find the optimal bandwidth for each system.

As supported by the research issues described in the previous section, it is impossible to implement an exact analysis and modeling of NCSs. In this dissertation, the following assumptions are made to simplify the analysis and modeling procedures throughout the research.

1) The sensor is a strict clock-driven device, and the controller and the actuator are strict event-driven devices. Sensor information is sent at every designated time, and controller outputs and actuator updates are implemented when the event is triggered.

2) The plant input disturbance and the sensor output noise are zero-mean white Gaussian noises (WGNs). They are independent of previous states, control inputs, and network-induced time delays.

3) The standard deviation of sensor output noise is small and fixed. Thus, the sensor noise effect can be included in the error variance of disturbance. When input disturbance is zero, the sensor output noise is one of the major sources of the output error.

4) All the sensor feedbacks or control outputs are sent in one packet for each iteration.

5) All four DC motors have identical properties and specifications. Thus, all motors show dynamically identical behavior.

2.2 Time Delays and Packet Dropouts in NCSs

Unlike a traditional point-to-point communication channel, NCSs introduce different forms of time delay uncertainties and packet dropouts between networks as shown in Fig. 2-1. These time delays and packet losses can originate from network-induced time delays as well as the processing time required for coding and computation. In the worst case, the time delays influence that the data packet does not arrive at the designated location within the sampling period. If a delayed packet arrives at a buffer that is already full, the packet is discarded, which may cause system instability.

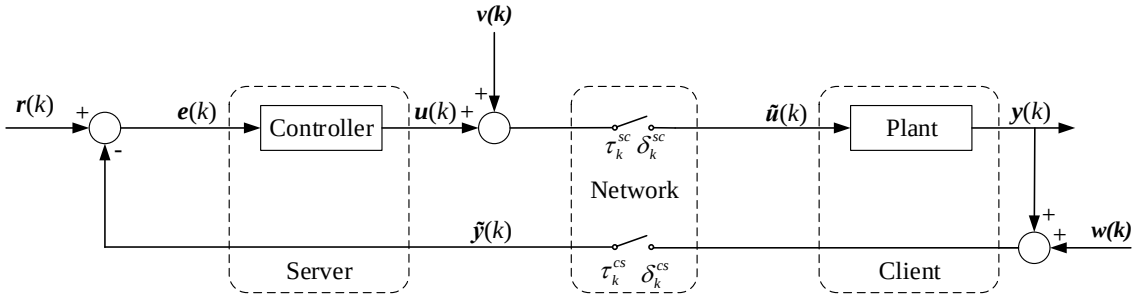

Figure 2-1. Block diagram of an NCS with disturbance, *v*(k), and noise, *w*(k). The independent random delays from server-to-client and client-to-server are represented as τ_k^{sc} and τ_k^{cs}, and the packet dropouts are denoted as δ_k^{sc} and δ_k^{cs}, respectively

NCSs have not only processing time delays, τ_p, but also network-induced time delays. Between them, the network-induced time delays can be generally described with five categories as follows.

i. Proceeding delay (T_{procd}): The required time to analyze a packet header and decide where to send the packet.
ii. Queuing delay (T_{queue}): The time a packet is enqueued until it is transmitted.
iii. Blocking delay (T_{block}): The time a message must wait once a node is ready to sent it.
iv. Transmission delay (T_{frame}): The time required to send all the bits in a packet on the transmission medium in use.
v. Propagation delay (T_{prop}): After the first bit is sent on to the transmission medium, the time required for the bit to propagate to the end of its physical trajectory.

Figure 2-2 presents the timing diagram during a single sampling period. When a message arrives at the server, the control task starts with a computing control output based on sensor outputs (T_{comp}) and encoding the data packet (T_{code}). Then, the router analyzes a packet header and decides where to send it (T_{procd}). After that, the data packet will be held in a queue waiting for transmission (T_{queue}) while other nodes are sending messages or resending them because of a message collision (T_{block}). When the network is idle, the data packet is transmitted to the client ($T_{prop} + T_{frame}$). Once the data packet arrives at the client's computer, it is first decoded (T_{deco}), a control iteration is performed with the hardware, and a new data packet is calculated (T_{comp}). After that, in a way similar to the server process, the data packet is encoded and sent to the server. Finally, the server receives the data packet and decodes the data to use for a new control iteration. Therefore, we can determine the total time delay as

$$\begin{aligned}\tau_k &= \tau_{sc} + \tau_{ca} + \tau_p \\ &= \underbrace{TS_{code} + TS_{procd} + TS_{queue} + TS_{block} + TS_{frame} + TS_{prop}}_{\tau_{sc}} \\ &\quad + \underbrace{TC_{deco} + TC_{code} + TC_{procd} + TC_{queue} + TC_{block} + TC_{frame} + TC_{prop}}_{\tau_{cs}}, \\ &\quad + \underbrace{TS_{comp} + TC_{comp}}_{\tau_p}\end{aligned} \tag{2.1}$$

where τ_k is total time delay at the instant *k*, τ_{sc} is time delay between server to client, τ_{ca} is time delay between client and actuator, τ_p is a processing time delay, *TS* is time delay at the server, and *TC* is time delay at the client.

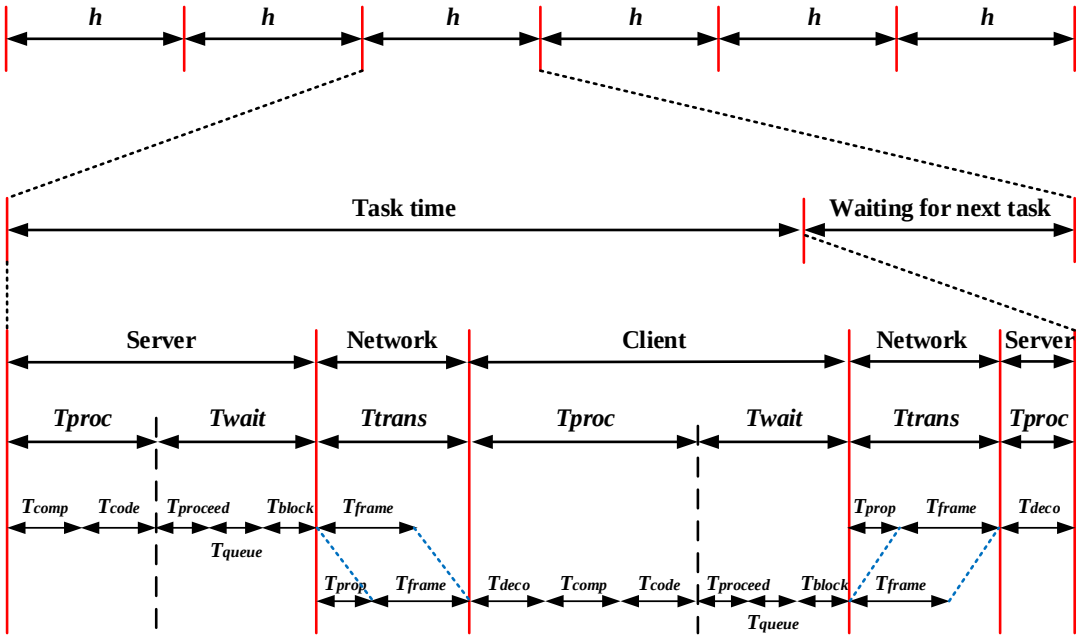

Figure 2-2. A timing diagram showing time spent sending a message from a server to a client for a sampling period

For the stability and performance of the system, each control task should be accomplished within a sampling period. Therefore, all the information of time delay should be studied before designing controller. Based on the total time delay, we should choose a sampling period within the allowable range to minimize the influence of time delays or packet dropouts. Thus, when NCSs are designed, an optimal bandwidth allocation method should be considered to keep the system stable.

2.3 Machine Learning Algorithms

Machine learning is a method of data analysis that automates analytical model building. There are three types of machine learning algorithms.

2.3.1 Supervised learning

The ouput for the given input is known in a supervised learning. It trys to model relationships between the target prediction output and the input features such that we can predict the output values for new data. Supervised learning algorithms include classification and regression. Examples of supervised learning are linear, nonlinear regression, decision tree, random forest, and ANN.

2.3.2 Unsupervised learning

The input data is given, but the outputs for given inputs are unknown in an unsupervised learning. Unlike supervised learning, unsupervised learning does not have a teacher. It is mainly used in pattern detection, grouping or clustering of data points. Examples of unsupervised learning are K-means clustering and Apriori algorithm.

2.3.3 Reinforcement learning

With reinforcement learning algorithm, the machine is trained to make specific decisions. The machine is exposed to an environment where it trains itself continually using trial and error. The algorithm interacts with environment to take actions which would maximize the reward. Q-learning and deep adversarial networks are the most popular reinforcement learning algorithms.

2.4 Dynamic Optimal Bandwidth Allocation for the Multi-Server Multi-Client System Using a Performance Index Function for the NCS System with Noise and Disturbance

2.4.1 Performance index functions

A PIF for an NCS gives a clear guideline with which to choose the optimal sampling periods for the system. Figure 2.3 displays the PIF comparison of different systems between a continuous-time control system, a discrete-time control system, and an NCS based on the sampling period [29]. When the control system specifications such as

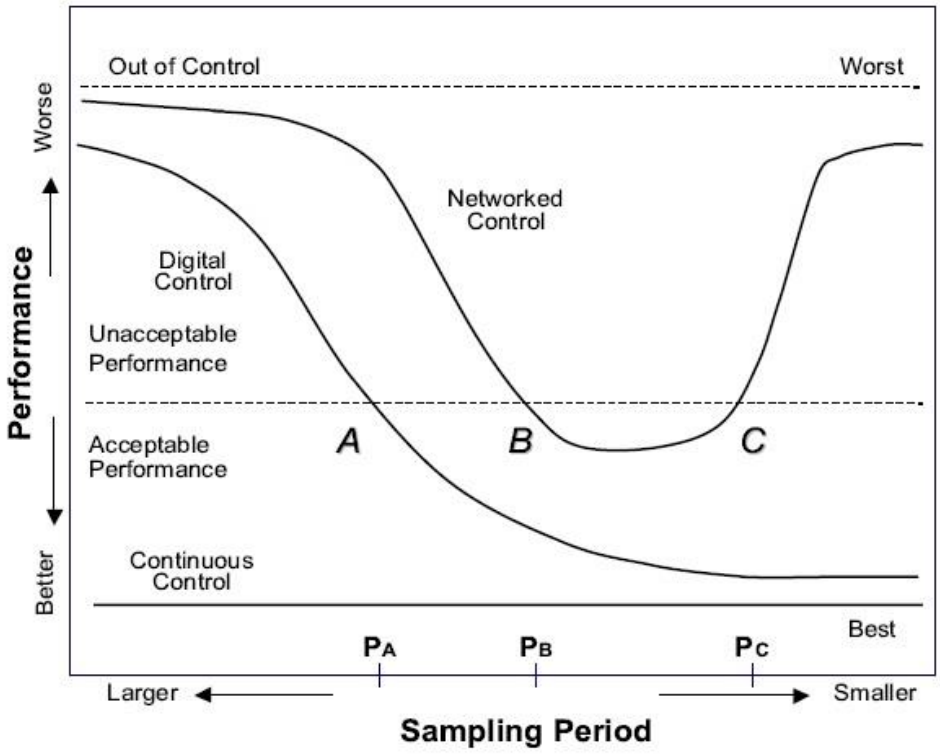

Figure 2-3. Performance index functions of continuous control, digital control, and networked control system, repectively, reprinted from [29]

overshoot, steady-state error, or phase margin are given, the acceptable, unacceptable, worst, and best sampling periods can be selected from the chart. Since the performance of the continuous-time control is independent of and not a function of the sampling period, the PIF of continuous control system is a constant and always presents the best results for the fixed control law. Considering a digital control system, however, the PIF depends on the sampling period assuming no other uncertainties. Here, at first the performance dramatically increases as the sampling period decreases, and it slowly increases and then converges to a constant value after a certain sampling period. Unlike in a conventional digital control system, a smaller sampling period (higher sampling frequency) does not always guarantee a better performance in an NCS. As a smaller sampling period induces heavier network traffic, the possibility of more time delays or packet dropouts also increases, and longer time delays finally result in degrading the system performance in the NCS. Thus, an optimal bandwidth allocation strategy is important to manage an NCS.

2.4.2 Mean squared error

A mean squared error (MSE) is adopted to measure the system performance in this dissertation because, unlike the integral absolute error (IAE) or the integral time-weighted absolute error (ITAE), the MSE is not a function of time and is useful for controlling the system with various disturbances and noise based on MSEs in real time. The MSE presents the mean of squared errors and penalizes a large error. This method was widely used in various research results [58], [59], and is formulated as follows:

$$MSE_i = \frac{1}{n}\sum_{k=k_0}^{k_f}(e_i^k)^2, \tag{2.2}$$

where n is the number of steps, k_0 and k_f are the initial and final times in the interval of interest, and e_i^k is the error of system i at time k.

For each individual plant, the MSE will have a different value when a sampling frequency is changed. The MSE will also have a distinctive value depending on the standard deviations of disturbance or noise. Thus, a set of accumulated MSEs of a system over disturbance or noise as well as sampling frequencies will be employed.

2.5 Artificial Neural Network

An ANN is an interconnected group of nodes, similar to the network of neurons in a human brain. In 1943 McCulloch and Pitts described how networks of artificial neurons can be made to perform logical functions [60]. Figure 2-4 shows the general diagram of an ANN composed of an input layer, two hidden layers, and an output layer. Each layer is made up of nodes. The output of a node in a certain layer becomes an input of a node in the next layer. A node receives inputs, changes their internal state according to those inputs, and produces outputs depending on the inputs, weights, and activation functions as below.

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right), \tag{2.3}$$

where y is the output of a node, w is a weight, x is an input, θ is a bias, and f is an activation function. The most popular types of activation functions are the sigmoid function and the hyperbolic tangent function.

An ANN is trained using experimental data and using an ANN, the standard deviation of input disturbance in an NCS can be estimated. After that, the optimal sampling frequency is calculated with another ANN. The inputs to estimate the standard deviation of disturbance are sampling period and the MSE of the system and the output is the estimated disturbance. To determine the optimal sampling frequency of each system, weight of each system, safety margin of BU, and estimated disturbance are inserted, and the optimal samping frequency is calculated as an output.

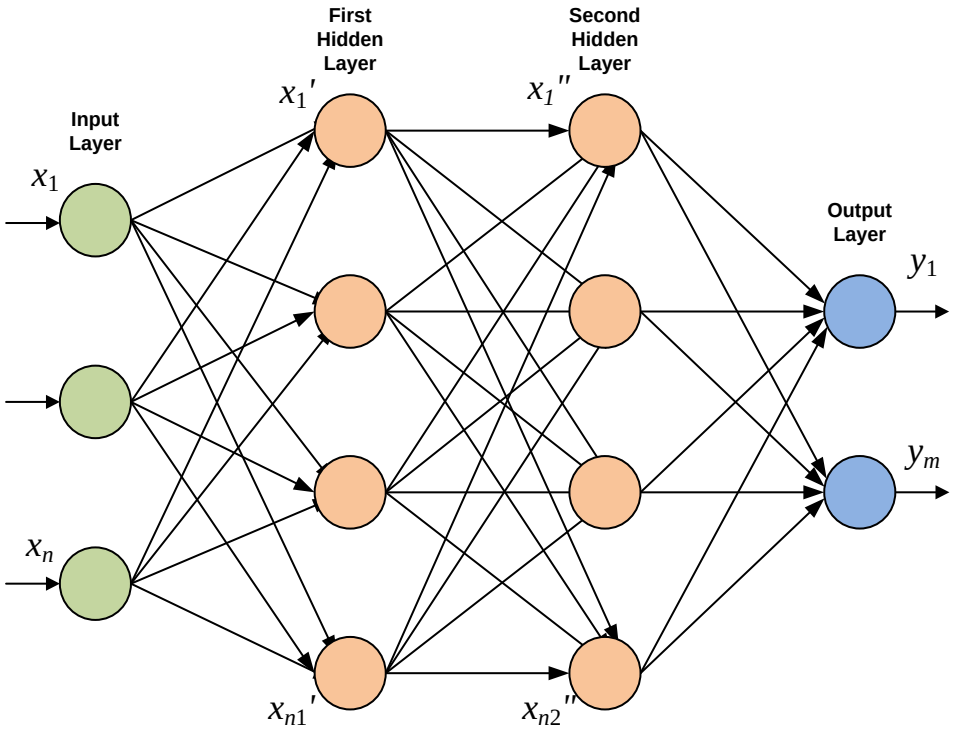

Figure 2-4. General diagram of an artificial neural network

2.6 Q-learning

Q-learning is a model-free reinforcement learning algorithm introduced by Watkins [61]. It includes an agent, a set of states *S*, and a set of actions *A*, per state. When the agent is in state *s*, it does action *a*, receives reward *r*, and goes into a new state *s′*. The goal of the agent is to maximize the total reward based on Q-values. The details of the terminology used in Q-learning are explained in Table 2.1.

Q-values are updated using the Bellman equation as shown in Eq. 2.4.

$$New\ Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)], \tag{2.4}$$

where *New Q*(*s*,*a*) is a new Q-value, *Q*(*s*,*a*) is a current Q-value, α is the learning rate, *R*(*s*,*a*) is a reward for taking action *a* at state *s*, γ is a discount factor, max*Q′* (*s′*,*a′*) is a

maximum expected future reward given the new s' and all possible actions at that new state.

The Q-learning algorithm can be processed as follows.

1. Arbitrarily initialize Q-values, ($Q(s,a)$)
2. Observe the current state s.
3. Choose an action a in the current state s based on the current Q-value using the epsilon greedy strategy. Generate a random number and if the number is greater than epsilon, then perform "exploitation." Otherwise, perform "exploration."
4. Take action a and observe the outcome in state s' and reward r.
5. Update the function $Q(s,a)$ according to Eq. 2.4.
6. Set the state to the new state and repeat the process until learning is completed or a terminal state is reached.

In an NCS, if estimated disturbance, weight of each system, and safety margin of BU are known in a current state, an optimal bandwidth can be determined by updating Q-values and choosing an action which has a maximum Q-value. Unlike the PIF or ANN method, the Q-learning does not need to design PIFs and to collect experimental data to find the optimal sampling frequency for each system.

Table 2.1. Description of terminology used in a Q-learning

Action (*a*)	An action a is the set of all possible actions the agent can make.
State (*s*)	A state s is an immediate situation in which the agent is located.
Reward (*r*)	A reward r is the feedback by which the success or failure of an agent's actions are measured.
Q-value (Q)	Q-value stands for the quality value of the action. Q-value maps state-action pairs to rewards.
Learning rate (α)	The learning rate α is set between 0 and 1. Setting it to 0 means that the Q-values are never updated, hence nothing is learned. Setting a high value such as 0.9 means that learning can occur quickly.
Discount factor (γ)	The discount factor γ determines the importance of future rewards. It is set between 0 and 1 and is designed to make future rewards worth less than immediate rewards.
Policy (π)	The policy is the strategy that the agent employs to determine the next action based on the current state. It maps states to actions which promise the highest reward.
ε-greedy	In Q-learning, the action with the highest Q-value is determined and is called the greedy action. To update Q-values, the greedy action (exploitation) is selected with probability (1- ε) and a random action (exploration) is selected with probability ε.

3. EXPERIMENTAL SETUP

3.1 Software Setup

An appropriate real-time operating system (OS) is required to minimize the time delay caused by data processing and communication, and to successfully implement a distributed control system via a network. Among commercially available OSs, Ji tested Windows, Linux, and Linux with RTAI, and found that Linux with RTAI gave a minimum time delay [62]. Linux Redhat 7.3 with Real-time Application Interface 3.4 (RTAI 3.4) runs on servers as the OS, and Linux Ubuntu 6.10 with RTAI 3.4 runs on clients 1, 2, 3, 4, and 5 as shown in Fig. 3-1. The control and measurement device interface (Comedi) is used as drivers and libraries of data acquisition on client computers. The programs for servers and clients 1, 2, 3, 4, 5 are developed using C language to control each system and communicate with each other. For wireless communication for client 6, a Dell Inspiron 1525 laptop computer, which has an Intel Core 2 Duo T7250 2.00 GHz processor and 4 GB RAM, was used where Microsoft Windows XP runs as the OS and programs were developed in Visual Basic 6.0 and Visual C++ 2008. Samba [63] was chosen as the Windows interoperability suite of programs for Linux as shown in Fig. 3-1, because Windows and Linux cannot communicate directly. Samba 5.0 was installed in a desktop computer running with Ubuntu 6.10 with RTAI 3.4 as an interoperability suite. Table 3.1 shows the specifications of server and client computers. The codes for servers, clients, and the interoperability suite are listed in Appendix A.

Table 3.1. Specifications of server and client computers

	Server 1	Server 2	Client 1	Client 2	Client 3	Client 4	Client 5	Gateway	Client 6
Processor	Celeron	Pen 4	Pen 4	Pen 3	Pen 3	Pen 4	Pen 4	Pen 4	Core 2 Duo
CPU Clock (Ghz)	0.6	1.5	1.7	1.0	1.0	1.7	1.7	1.5	2.0
Memory (MB)	128	256	256	256	256	384	512	256	4000

3.2 Hardware Setup

Figure 3-1 presents the hardware setup for multi-server and multi-client experiments. As shown in Fig. 3-1, an NCS in this research consists of two servers and six clients (a ball maglev system, four DC motor systems, and a wheelchair robot). The server

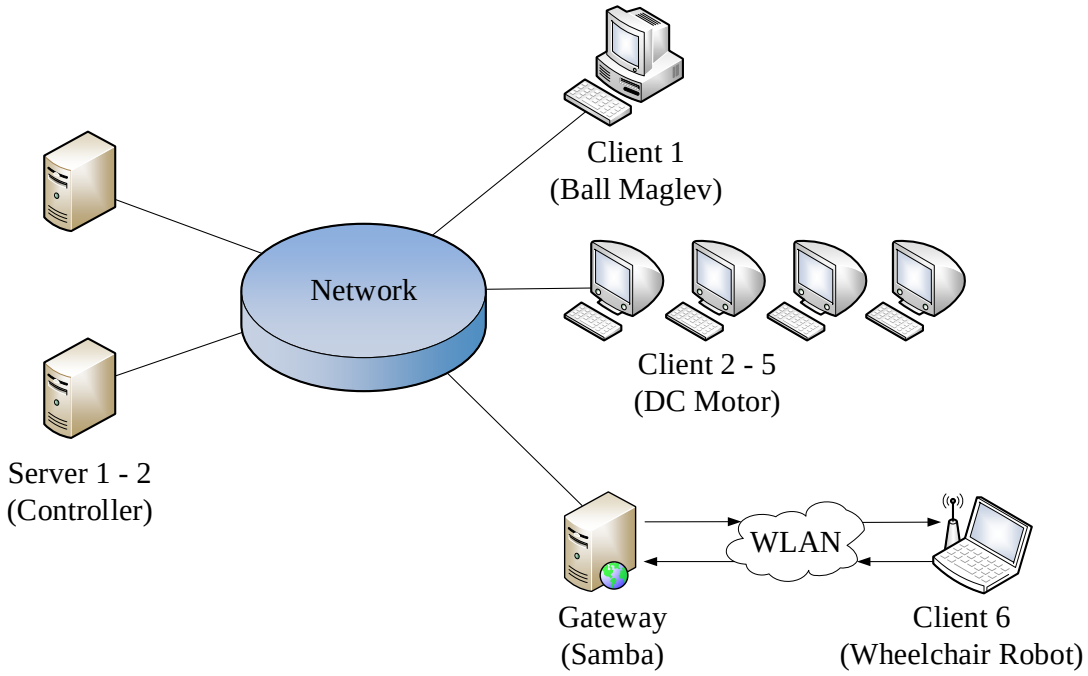

Figure 3-1. Experimental setup of an NCS

PCs calculate control inputs after receiving sensor output data from clients over the network and sends them to each client's PC over the network. Each client sends control input to the hardware plant, receives sensor data from the plant, and sends those data to the server PCs. A PCI-6221 data-acquisition card manufactured by National Instruments (NI) allows the hardware test bed to send out sensor data and receive control data through the local area network (LAN) or the wireless local area network (WLAN). Clients 1 to 5 represent clients wired into the NCS and client 6 represents a wireless client.

3.3 Ball Maglev System

Figure 3-2 presents a steel ball maglev system as Client 1 [64]. An electromagnet in the system allows the steel ball to be levitated at a predefined steady-state equilibrium position. Two personal computers (PC), one for server and one for a client, a position sensor, and a pulse-width modulator (PWM) power amplifier are used to levitate the ball.

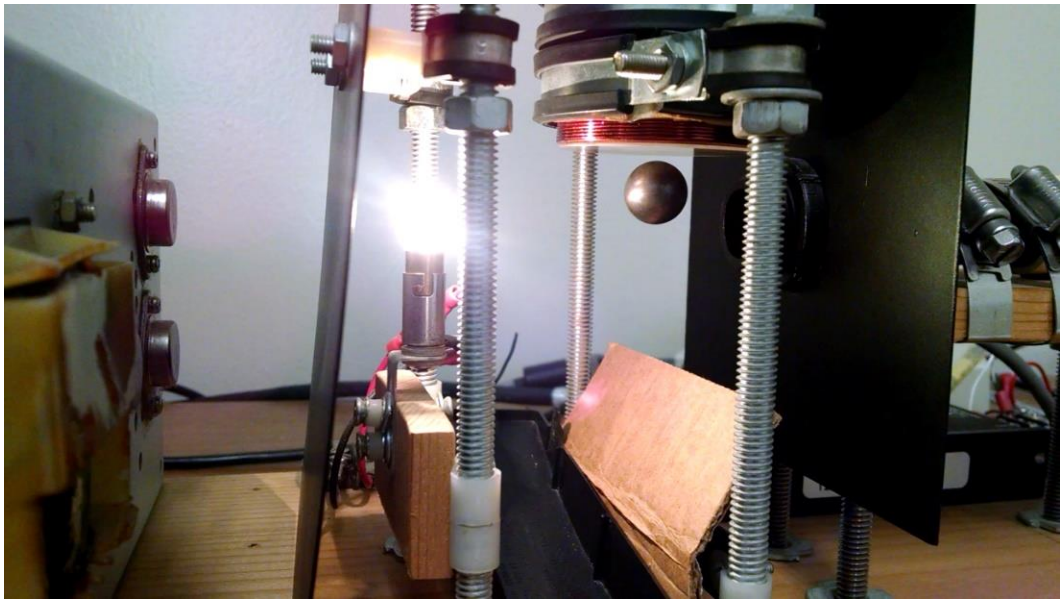

Figure 3-2. Ball maglev system

The optical position sensor unit consists of an incandescent light bulb as a light source, a CdS photocell, and a 15-V DC power supply.

The transfer function of a ball maglev system is [64]

$$G(s) = \frac{0.02792}{0.0086s^2}. \tag{3.1}$$

A lead-lag controller is used to stabilize the system as [64]

$$D(s) = \frac{33300s^2 + 2.564\times10^6 s + 1.632\times10^7}{s^2 + 700.7s + 490}, \tag{3.2}$$

where the input is the voltage to the electromagnet model, and the output is the position of the steel ball with a unit of mm.

A lead-lag compensator is chosen to control the maglev system because a lead compensator speeds up a response by lowering the rise time and decreases the transient overshoot, and lag compensator improves the steady-state accuracy of the system. As shown in Eq. 3.1, the open-loop transfer function of ball maglev system is marginally stable and the system requires a fast speed sampling period to obtain stability and performance. Therefore, a 3-ms sampling period was chosen for the system [64].

Figure 3-3 presents an experimental result for the ball maglev system with a lead-lag compensator when the reference position is 4 mm for 60 seconds. The system kept a 4 mm reference position after 10 seconds.

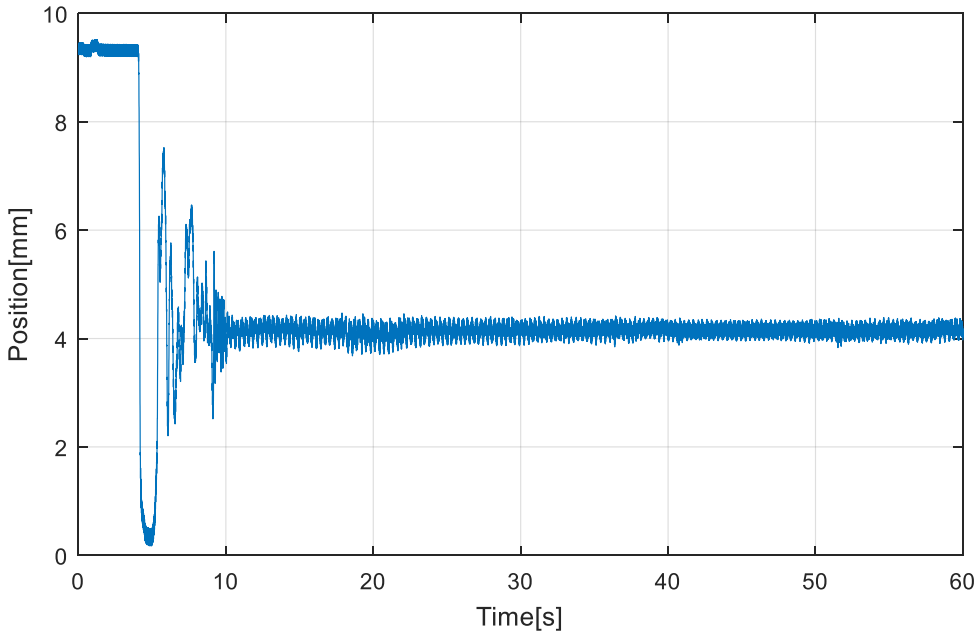

Figure 3-3. The position of a steel ball in the ball maglev system when the reference is 4mm

3.4 DC Motor System

Clients 2 to 5 are on the DC motor speed control system shown in Fig. 3-4 [65]. The speed control is achieved by controlling the output voltage of a pulse-width modulation (PWM) amplifier. Encoders located at the bottom of each motor measure the angular displacement of the motor shaft per unit of time. A PCI-6221 data-acquisition (DAQ) card by NI enables the test bed to send out sensor-output data packets and receive control-input data packets through the LAN.

The transfer function of a DC motor system is

$$G(s) = \frac{20.2}{9.92s + 2.57}. \tag{3.3}$$

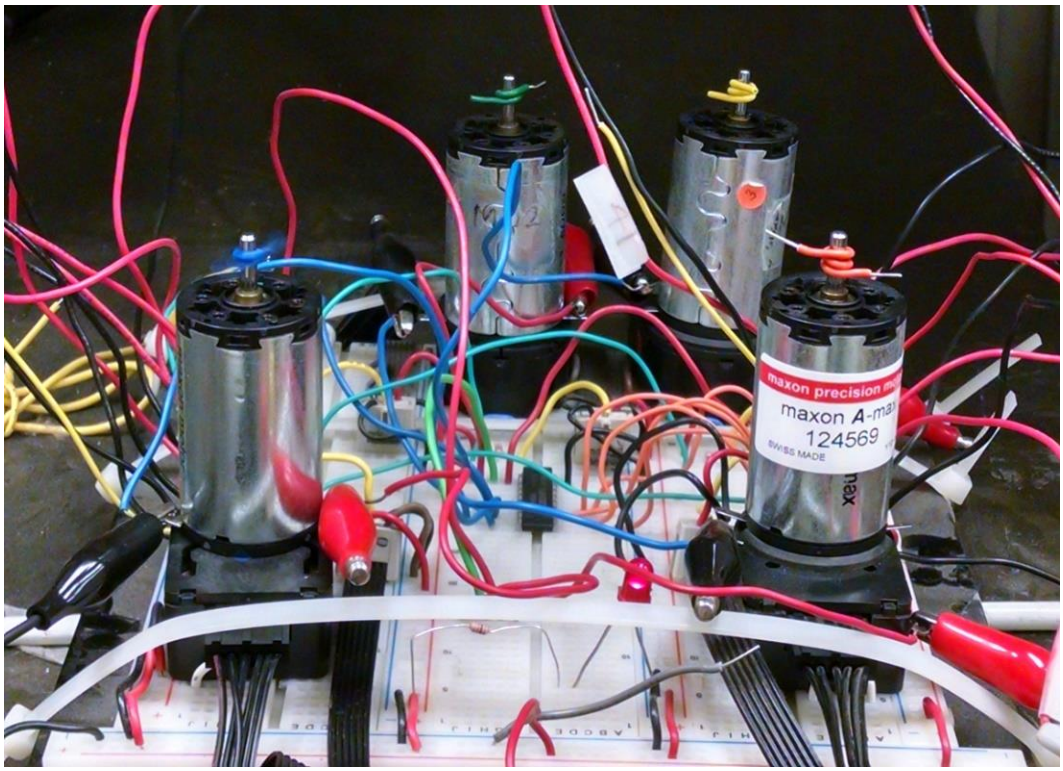

Figure 3-4. A DC motor system

A proportional-integral (PI) controller is used to control the DC motor as

$$D(s) = \frac{1.8s + 6}{s}, \tag{3.4}$$

where the input is the armature voltage, and the output is the speed of the DC motor with a unit of revolutions per second (rps).

A PI compensator is chosen to control the DC motor system since a derivative compensator is subject to high frequency noise or disturbance such as our test bed. Coefficients of compensator are selected using root locus, where $\zeta = 0.562$, $M_p = 11.9\%$, $\omega_n = 3.49$ rad/s, $t_r = 0.52$ s, and $t_s = 2.35$ s as presented in Fig. 3.5. As

shown in Eq. 3.3, the system is open-loop stable and requires medium speed sampling period. Therefore, various sampling periods, from 3 ms to 30 ms, will be applied for each DC motor system to determine optimal periods. Figure 3-6 shows a Simulink model of the DC motor system and its simulation results are shown in Fig. 3-7. Figure 3-8 represents an initial experimental result of the DC motor system without disturbance or noise using the PI controller with a sampling period of 3ms and a reference speed of 7 rps for 3 seconds. The DC motor system follows reference input quite well without disturbance or noise.

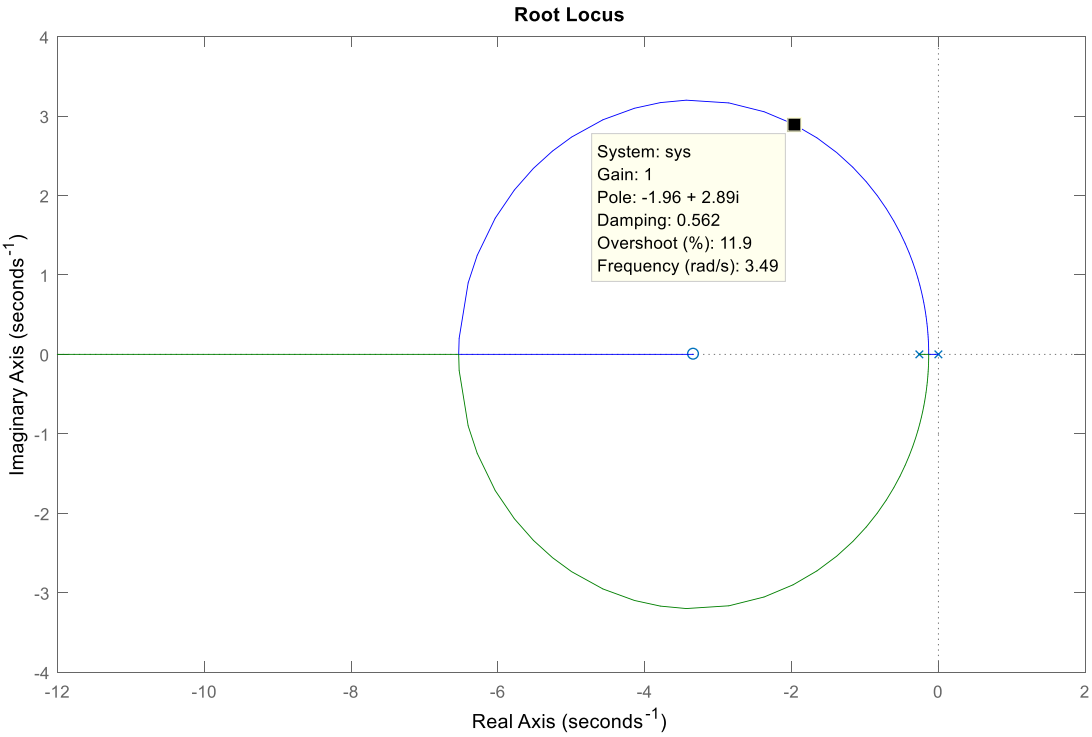

Figure 3-5. Root locus to determine coefficients of compensator for the DC motor system

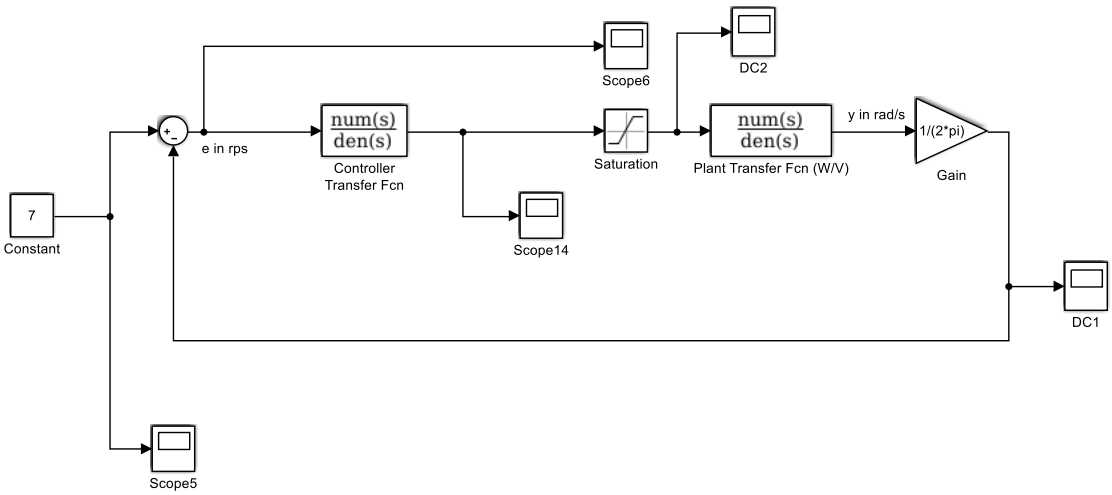

Figure 3-6. Simulink model of a DC motor system

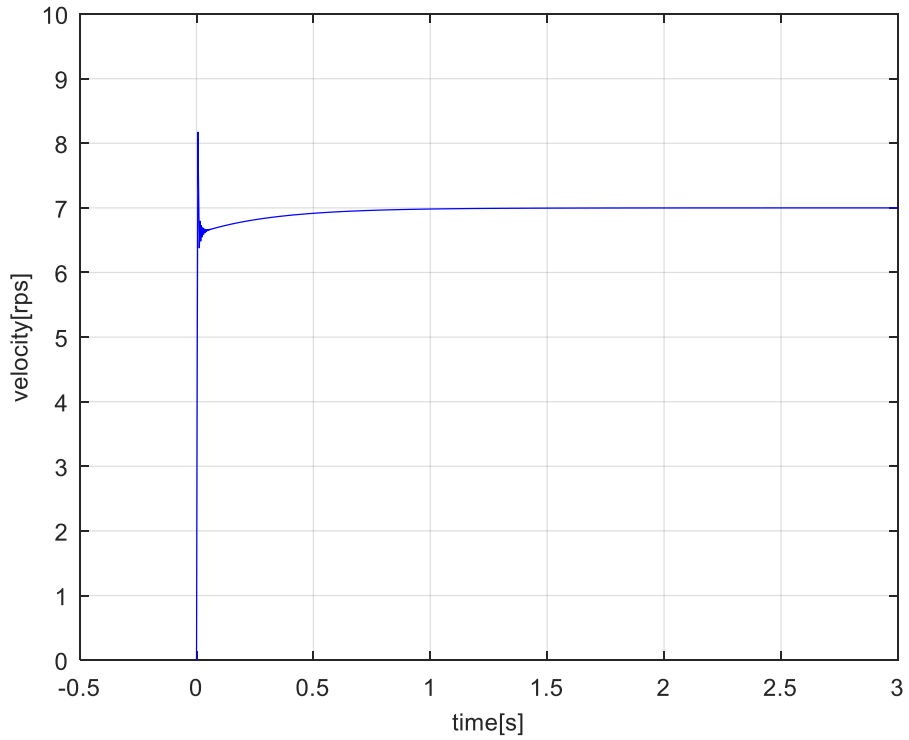

Figure 3-7. DC motor control result with a 7-rps reference input using Simulink

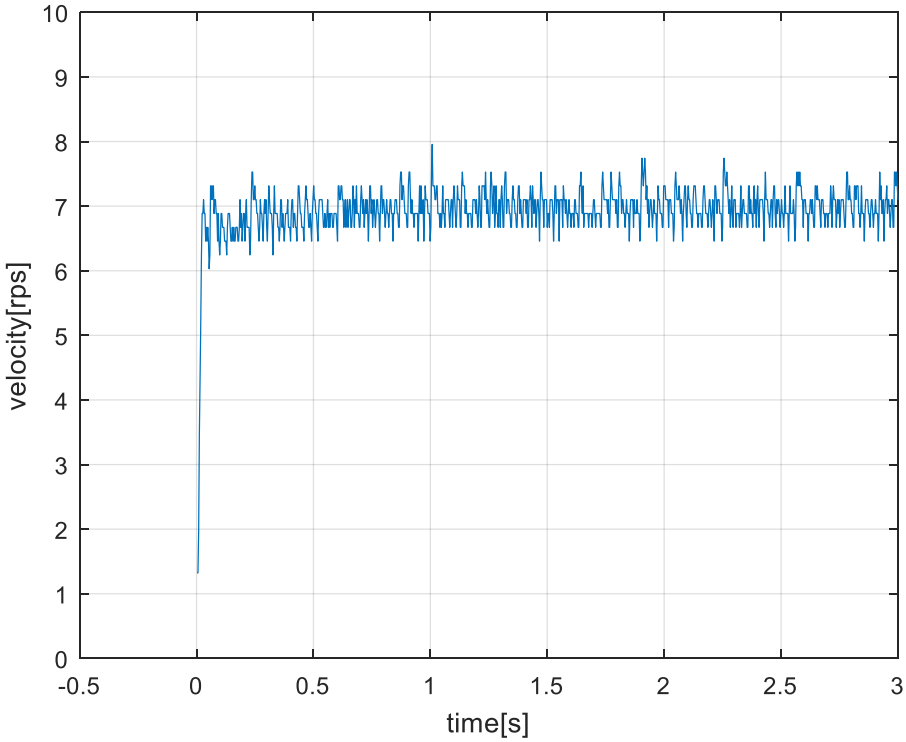

Figure 3-8. The speed of a DC motor when the reference speed is 7 rps

3.5 Wireless Autonomous Wheelchair Robot System

Figure 3-9 shows the autonomous wireless wheelchair robot based on the frame of an Invacare Ranger TM II electric powered wheelchair [66]. Two 12-V DC motors independently drive the two front wheels and two small rear wheels follow the front wheels. The speed of the motors is controlled by the output voltage of the PWM amplifiers on the board of two Diverse Electronic's modular MC-7 motor controllers. All data acquisition and control data packet exchanges are performed by an NI USB-6501 DAQ card. In order to measure distance from obstacles in the path, three Sharp GP2D15 and two Sharp GP2D12 infrared distance-measuring sensors are mounted on a sensor bracket located in front of the wheelchair. The moving distance of each wheel is calculated by

measuring the number of revolutions of the motor shaft, which is detected by the Hall-effect sensors mounted on the rear casing of both motors and the data is fed to a 74HC191 counter chip.

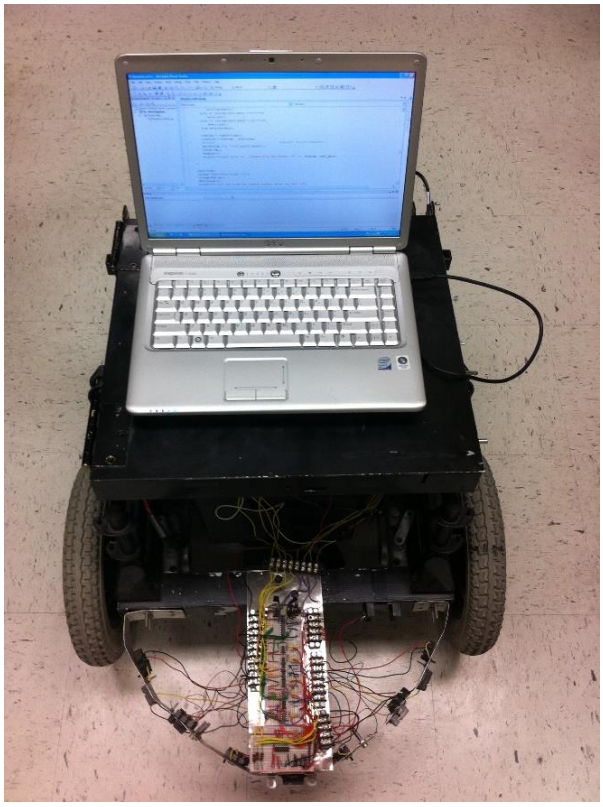

Figure 3-9. Autonomous wireless wheelchair robot

This system moves 0.21 m/s and rotates 0.733 rad/s. Thus, it does not require a fast speed speed sampling period. Figure 3-10 presents an experimental result of the wheelchair robot to avoid obstacles when path is clear. The robot moved straight at a 300 ms sampling period for 30 seconds.

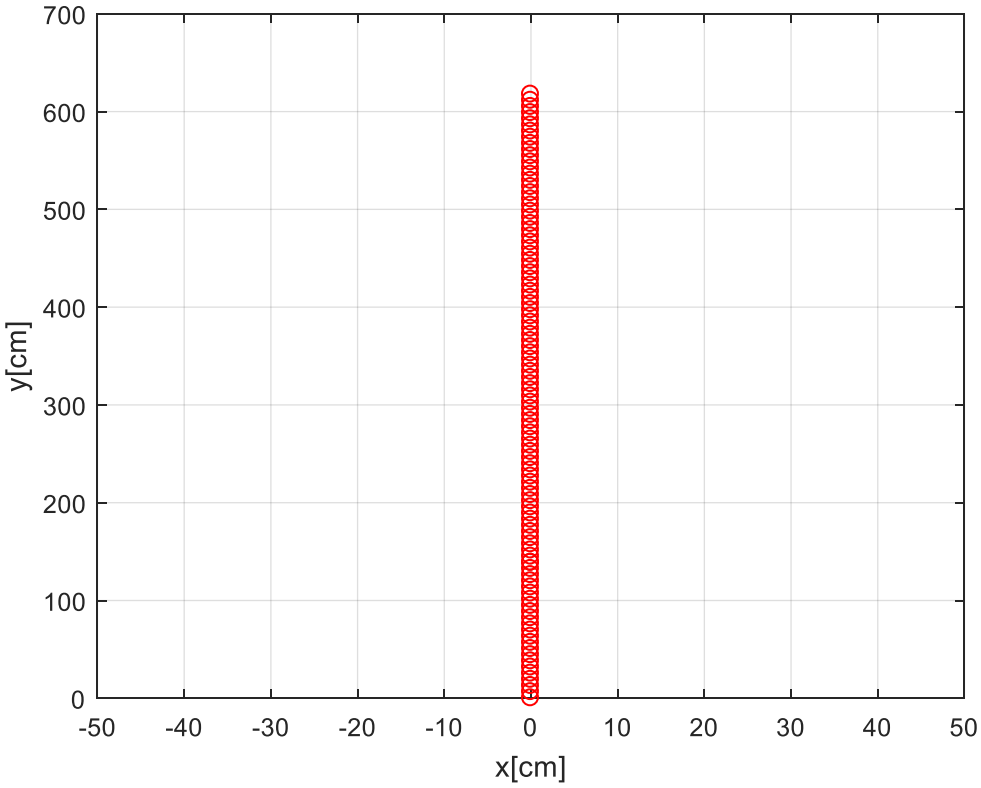

Figure 3-10. Straight motion of the wheelchair robot

3.6 Network Bandwidth

According to Ji and Kim [67], the relationship between the sampling periods and bandwidth utilization (BU) for each client in the NCS can be expressed as

$$b_i^k = \frac{\tau_i^k}{h_i^k} = \tau_i^k f_i^k , \tag{3.5}$$

where b_i^k is the BU, h_i^k is the sampling period, f_i^k is the sampling frequency, and τ_i^k is the total time delay. The subscript i indicates the index of the clients in the NCS, and the superscript *k* indicates the *k*-th iteration of the loop. Then BU represents a portion of the

network bandwidth assigned to client at the control iteration *i*. From Eq. 3.5, given a certain amount of time delays, a small BU implies a large sampling period (low sampling frequency) and more bandwidth available for other functions and control purposes in the same network. If the BU approaches the network bandwidth saturation threshold, the network will be overloaded and create more time delays or packet dropouts.

After PING tests were performed for 100 round trips, the round-trip time between the server 1 and client 1, the ball maglev system, is calculated by averaging them. The average round-trip time from the server 1 to client 1 is 0.388 ms with a standard deviation of 0.011, the round-trip time from client 1 to the server 1 is 0.393 ms with a standard deviation of 0.024, and the data processing time is 0.915 ms. For client 2, the DC motor system, the average round-trip time from the server 1 to client 2 is 0.378 ms with a standard deviation of 0.012, the round-trip time from client 2 is to server 1 is 0.397 ms with a standard deviation of 0.027, and the data processing time is 0.866 ms. Client 6, the wheelchair robot system, takes 0.391 ms from gateway to server 1 with a standard deviation of 0.013, 0.373 ms from server 1 to client with a standard deviation of 0.018, 2.01 ms from client to gateway with a standard deviation of 0.8819, 1.97 ms from gateway to client with a standard deviation of 0.9370 for the average round-tip time, and 0.738 ms to process. Client 6 should count the average round-trip time between client 6 and gateway as well as between gateway and server because client 6 comunicates with the server 1 through WLAN and gateway. Therefore, the total time delay can be calculated by summing the one-way trip time from server to client, from client to server, and the data processing time as Eq. 3.6 to 3.8.

$$\tau_1^k = (0.388 + 0.393)/2 + 0.915 = 1.31 \text{ ms}. \tag{3.6}$$

$$\tau_2^k = (0.378 + 0.397)/2 + 0.866 = 1.25 \text{ ms}. \tag{3.7}$$

$$\tau_6^k = (0.391 + 0.373 + 2.01 + 1.97)/2 + 0.738 = 3.11 \text{ ms}. \tag{3.8}$$

Table 3.2 presents the BU examples for the NCS for the ball maglev system, DC motor system, and wheelchair robot system with various sampling periods. If the total BU exceeds a certain amount, one hundred percent, the network induces a time delay or packet dropout. Thus, when the optimal bandwidth allocation algorithm for the multi-server and multi-client system is designed, the BU should be considered.

Table 3.2. BU for the NCS for the ball maglev system and the DC motor system with various sampling times

Sampling period / Clients	2.5 ms	3 ms	4 ms	6 ms	9 ms
Client 1 (Ball maglev)	52%	44%	33%	22%	15%
Client 2 (DC motor)	50%	42%	31%	21%	14%
Sampling period / Clients	100 ms	150 ms	200 ms	250 ms	300 ms
Client 6 (Wheelchair)	3.12%	2.08%	1.56%	1.25%	1.04%

4. PERFORMANCE INDEX FUNCTION FOR THE NCS WITH DISTURBANCE AND NOISE*

4.1 Performance Index Function for the NCS

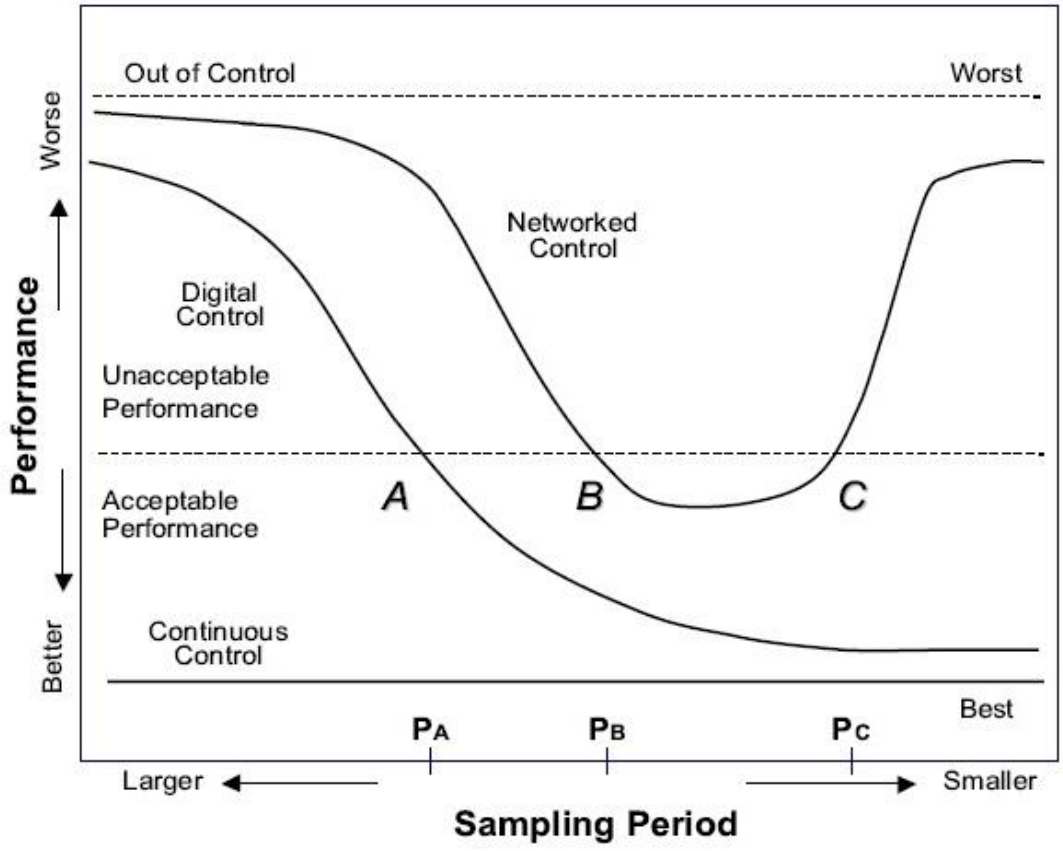

Figure 4-1. Performance index functions for continuous control, digital control, and networked control systems, respectively, reprinted from [29]

*Part of this section is reprinted with permission from "Performance-Index Functions in Networked Control Systems with Disturbance and Noise" by Kuktae Kim and Won-jong Kim, *ASME* 2015 *International Mechanical Engineering Congress and Exposition*, vol. 4B, pp. V04BT04A020, Copyright 2015 by ASME.

Figure 4-1 presents the PIFs to compare the performance of different systems between a continuous-time control system, a discrete-time control system, and an NCS based on the sampling period. A continuous control always presents best performance and a digital control shows better performance when sampling period becomes smaller. Unlike a digital control system, a smaller sampling period (higher sampling frequency) does not always guarantee a better performance in the NCS. As a smaller sampling period induces a heavier network traffic, the possibility of more time delays or packet dropouts also increases, and longer time delays finally result in a degradation of the system performance in the NCS. Therefore, when designing an NCS, bandwidth and sampling period should be considered and a PIF for an NCS can give clear guidance to determine the optimal sampling frequency for a single system. If the PIFs are integrated with other optimal algorithms such as nonlinear constrained optimization, ANN, and Q-learning, the optimal bandwidth allocation method for a multi-server and multi-client system can be determined.

4.2 Exponential and Polynomial Approximations of a PIF

4.2.1 Exponential and 4^{th}-order polynomial approximations of a PIF with fixed standard deviation of disturbance, which is a function of frequency

Unlike Dong and Kim's study [31], which modeled an NCS with time delays, in this dissertation, exponential and 4^{th}-order polynomial approximations of PIFs are proposed to model a relationship between disturbance and sampling frequency of the DC motor system in the NCS based on experimental data. The error variances measured from the experiments will be defined as a piecewise function of the sampling frequency, when

the standard deviation of disturbance or noise is fixed. Based on Fig. 4-1 and experimental data, exponential and polynomial approximations were used as PIFs. We can approximate an exponential PIF as below,

$$P_i(f_i^{\,k}) = \alpha_i e^{\beta_i f_i^{\,k}} + \gamma_i e^{\delta_i f_i^{\,k}}, \tag{4.1}$$

where P_i is the performance index of system i, $f_i^{\,k}$ is the sampling frequency of system i at iteration k , and α_i, β_i, γ_i, and δ_i are approximation coefficients.

The exponential approximation can closely follow the performance index of the system. However, the exponential function takes much more time to calculate and is not easy to obtain in real time compared to a polynomial function. A 2nd-order polynomial approximation has bigger errors compared to the exponential and 4th-order polynomial approximations. Thus, a 4th-order polynomial approximation is proposed as a PIF. For each individual plant, the general 4th-order polynomial PIF can be defined as

$$P_i(f_i^{\,k}) = a_i(f_i^{\,k})^4 + b_i(f_i^{\,k})^3 + c_i(f_i^{\,k})^2 + d_i f_i^{\,k} + e_i, \tag{4.2}$$

$$P_i(f_i^{\,k}) = p_{1i}(f_i^{\,k} - p_{2i})^4 + p_{3i}, \tag{4.3}$$

where a_i , b_i , c_i , d_i , e_i , p_{1i} , p_{2i} and p_{3i} are the approximation coefficients. A special form of the 4th-order polynomial, Eq. 4.3, as well as a general form of the 4th-order polynomial, Eq. 4.2, were introduced because the special form approximates experimental data better than the general form when the standard deviation of disturbance was less than 0.1333 and prevents the PIFs from having negative error variances, which sometimes takes place with a 4th-order polynomial.

Since P_i is the performance index of the system, we can find the optimal sampling frequency by calculating the frequency that minimizes the PIF, P_i, for a single-client system. The allowable frequency range for the client is also determined based on the PIF equations, when required performance is given.

4.2.2 2nd-order polynomial approximations of a PIF with a fixed sampling frequency, which is a function of standard deviation of disturbance

A 2nd-order polynomial approximation was selected for the PIF of the DC motor system with a fixed sampling frequency in terms of maximum disturbance based on experimental data. For each individual plant, the 2nd-order polynomial PIF can be defined as

$$P_i(d_i^k) = a_i(d_i^k)^2 + b_i(d_i^k) + c_i, \tag{4.4}$$

where d_i^k is the maximum disturbance of system i at iteration k and a_i, b_i, and c_i are the approximation coefficients. Once the PIF, a function of maximum disturbance, is obtained, the maximum disturbance can be estimated when the sampling frequency and error variance are informed.

4.2.3 6th-degree polynomial approximation for a 3-D PIF in terms of frequency and maximum disturbance

To describe a relationship between a PIF and a sampling frequency, a 4th-order polynomial is used and to present a relationship between a PIF and the maximum disturbance, a 2nd-order polynomial is used. Thus, a 6th-degree polynomial as shown in

Eq. 4.5 is proposed for a 3-D PIF to describe relationships between maximum disturbance, sampling frequency, and system performance.

$$\begin{aligned} P_i(f_i^k, d_i^k) = {} & p_{00} + p_{10}(f_i^k) + p_{01}(d_i^k) + p_{20}(f_i^k)^2 + p_{11}(f_i^k)(d_i^k) + p_{02}(d_i^k)^2 \\ & + p_{30}(f_i^k)^3 + p_{21}(f_i^k)^2(d_i^k) + p_{12}(f_i^k)(d_i^k)^2 + p_{40}(f_i^k)^4 + p_{31}(f_i^k)^3(d_i^k), \\ & + p_{22}(f_i^k)^2(d_i^k)^2 \end{aligned} \tag{4.5}$$

where f_i^k is the sampling frequency of system *i* at iteration *k*, d_i^k is the maximum disturbance of system *i* at iteration *k*, p_{00}, p_{10}, p_{01}, p_{20}, p_{11}, p_{02}, p_{30}, p_{21}, p_{12}, p_{40}, p_{31}, and p_{22} are the approximation coefficients.

4.3 Performance Index Function for the DC Motor System

4.3.1 Performance index function for the DC motor system in terms of frequency with fixed standard deviations of disturbances

Table 4.1 shows the average experimental results of the error variance between reference inputs (7 rps) and outputs for the DC motor system with various standard deviations of disturbances and sampling periods. Experiments were conducted on three different days and times to avoid specific unknown errors and find representative values. Each experimental data are presented in the Appendix B. Various standard deviations of white Gaussian disturbances were injected into the input voltage right after the server PC calculated a control input for the DC motor, and data including uncertainty was sent to the client PC to actuate the DC motor. The maximum range of disturbance was ±0 V, ±0.2 V, ±0.4 V, ±0.8 V, ±1 V, ±1.5 V, ±2 V, ±3 V, ±4 V, ±6 V, respectively. In this dissertation,

all disturbances are white Gaussian disturbance and it shows the Gaussian distribution that 99.7% of the data are within three standard deviations, σ, of the mean. Thus, one third of the maximum disturbance is standard deviation of disturbance. A small standard deviation of white sensor noise was assumed to exist and this results in errors when the disturbance is zero. The experimental results of the error variance in various sampling periods are also visually presented in Fig. 4-2 and a reduced version is shown in Fig. 4-3 to magnify the details of the behavior of a disturbance less than maximum 1.5 V. The various trends of the system performance are shown when various maximum disturbances are injected.

Table 4.1 Error variances between reference inputs (7 rps) and outputs with various standard deviations of disturbances

Sampling T.(ms) / Disturbance (m, σ^2)	2.5ms	3ms	4ms	5 ms	6 ms
$(0,0^2)$	0.076	0.058	0.063	0.086	0.133
$(0,0.067^2)$	0.174	0.120	0.116	0.134	0.158
$(0,0.134^2)$	0.408	0.313	0.257	0.245	0.245
$(0,0.268^2)$	1.297	1.068	0.828	0.699	0.580
$(0,0.333^2)$	1.912	1.610	1.236	1.029	0.826
$(0,0.5^2)$	4.066	3.450	2.669	2.184	1.698
$(0,0.667^2)$	6.802	5.828	4.606	3.775	2.904
$(0,1^2)$	12.475	10.971	9.052	7.723	6.155
$(0,1.333^2)$	17.215	15.611	13.387	11.873	9.744
$(0,2^2)$	23.851	22.101	20.260	18.228	16.300
Sampling T. (ms) / Disturbance (m, σ^2)	9ms	12ms	15ms	21 ms	30 ms
$(0,0^2)$	0.275	0.497	0.837	1.651	3.383
$(0,0.067^2)$	0.296	0.528	0.821	1.634	3.297
$(0,0.134^2)$	0.360	0.561	0.869	1.698	3.287
$(0,0.268^2)$	0.607	0.740	1.001	1.770	3.386
$(0,0.333^2)$	0.786	0.867	1.100	1.854	3.577
$(0,0.5^2)$	1.411	1.340	1.458	2.099	3.594
$(0,0.667^2)$	2.307	1.978	1.961	2.454	3.964
$(0,1^2)$	4.746	3.816	3.437	3.449	4.601
$(0,1.333^2)$	7.842	6.293	5.355	4.791	5.463
$(0,2^2)$	14.000	11.795	10.195	8.456	7.914

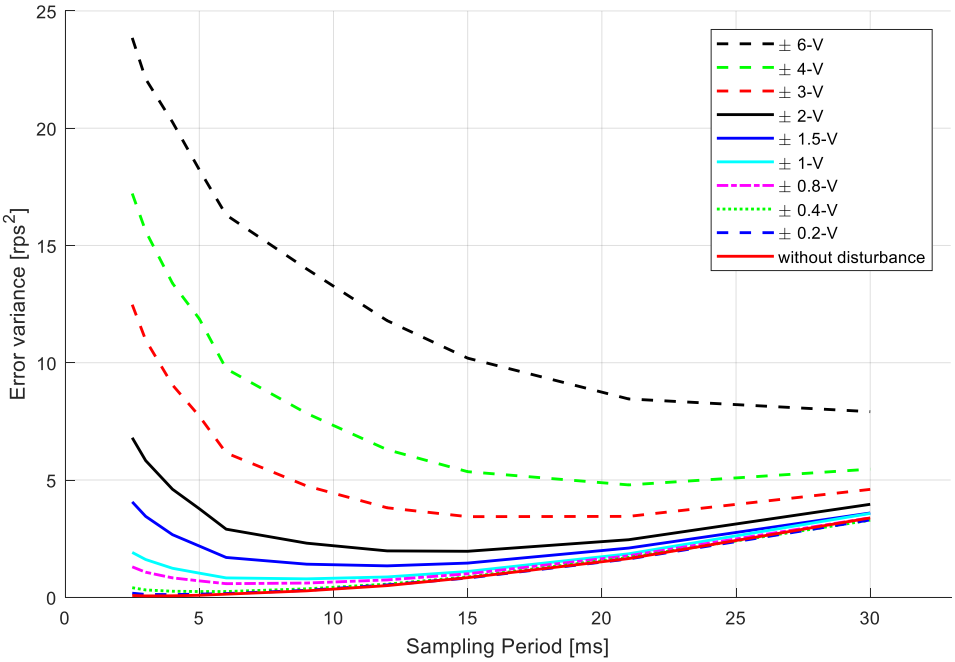

Figure 4-2. Error variances of DC1 with disturbances at the actuator at various sampling periods

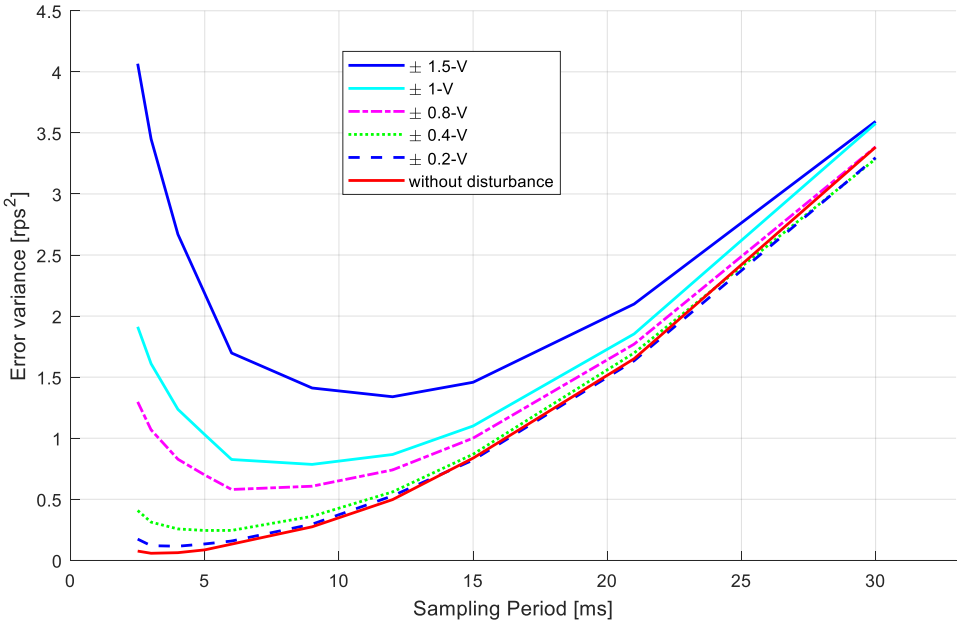

Figure 4-3. Reduced version of error variances of DC1 with disturbances at the actuator at various sampling times

Without the disturbance, the error variance continues to decrease when the sampling frequency increases (sampling period decreases). However, when the standard deviation of the disturbance is increased, the error variance decreases at a low frequency (slow sampling period) from the beginning and then increases after a certain sampling frequency.

Figs. 4-4 to 4-13 presents experimental data and PIFs using the 4^{th}-order polynomial and exponential approximations of a DC motor system with various sampling periods and a fixed maximum disturbance. To calculate the coefficients of each approximation, MATLAB cftool is used. A trust-region algorithm is used for exponential approximations and a linear least squares algorithm is used for the 4^{th}-order polynomial approximations. Approximation coefficients and R-squared values for each figure are listed in Tables 4.2 to 4.4. Standard deviations, σ, of disturbances are also included in Tables 4.2 to 4.4. When the 3σ of injected disturbance is within the interval of ± 1.5 V, the exponential pproximation presents a better R-squared value than do the 4^{th}-order polynomials and if the 3σ of injected disturbance is outside the interval of ± 2 V, the 4^{th}-order polynomial shows a better R-squared values. Both approximation methods have reasonable R-squared values which are higher than 0.93. Since 4^{th}-order polynomial approximations need less calculation time than exponential approximations, in this research 4^{th}-order polynomial approximations are used to determine optimal bandwidth allocation for the DC motor system.

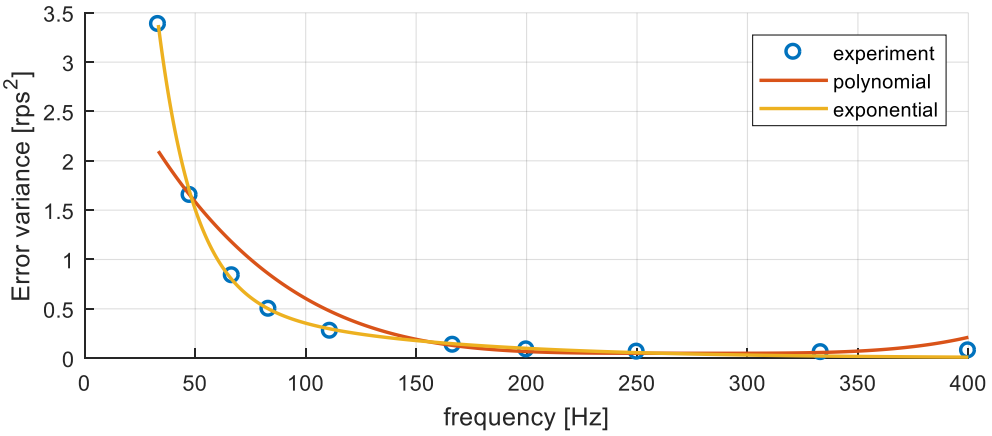

Figure 4-4. PIFs using 4th-order polynomial and exponential approximations of a DC motor system without disturbance at the actuator

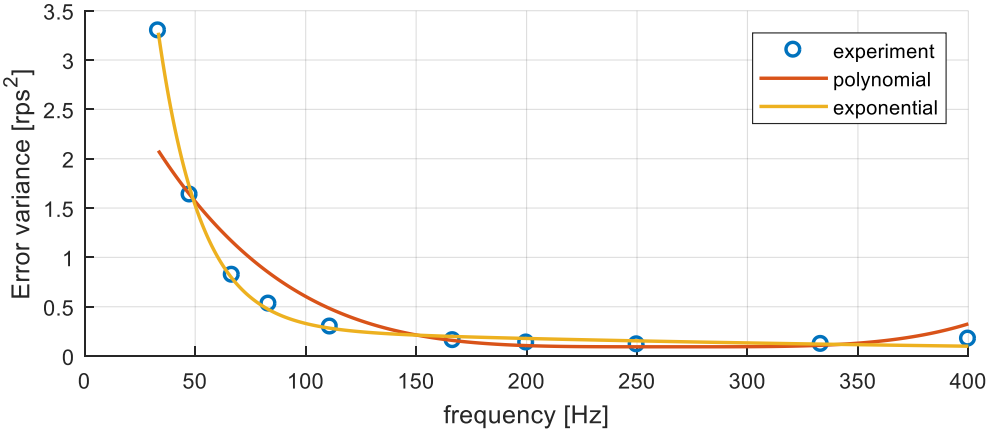

Figure 4-5. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±0.2-V disturbance at the actuator

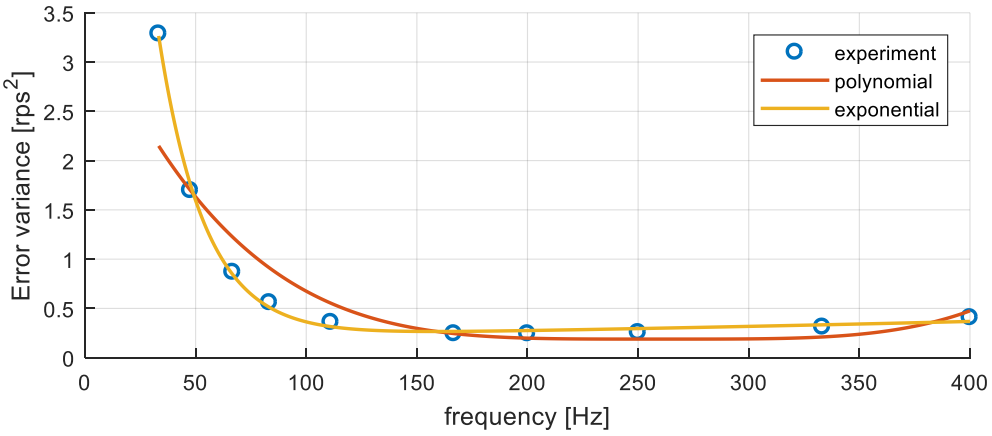

Figure 4-6. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±0.4-V disturbance at the actuator

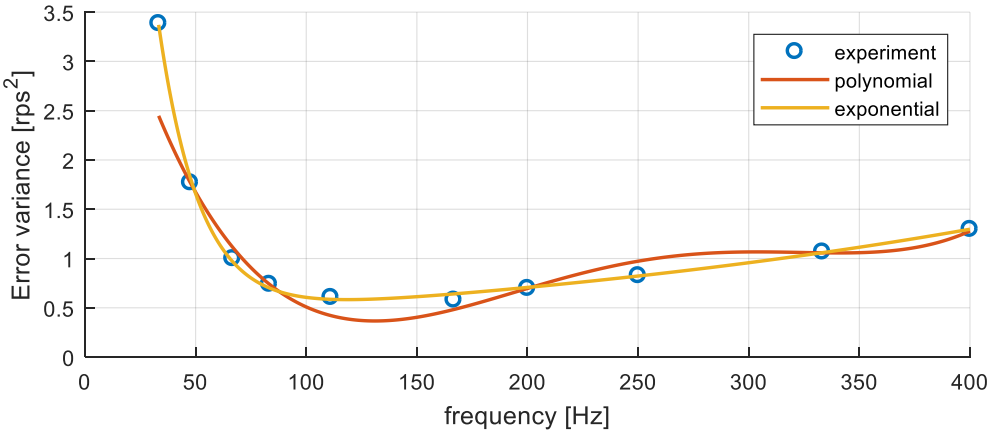

Figure 4-7. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±0.8-V disturbance at the actuator

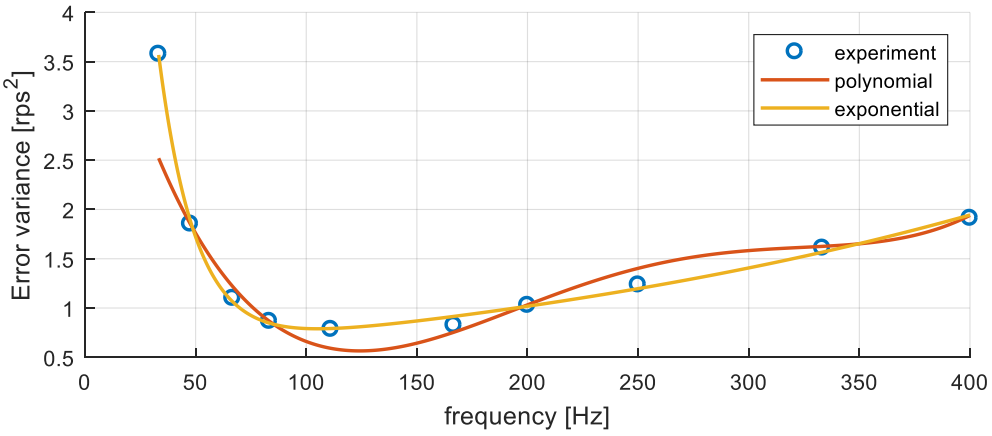

Figure 4-8. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±1-V disturbance at the actuator

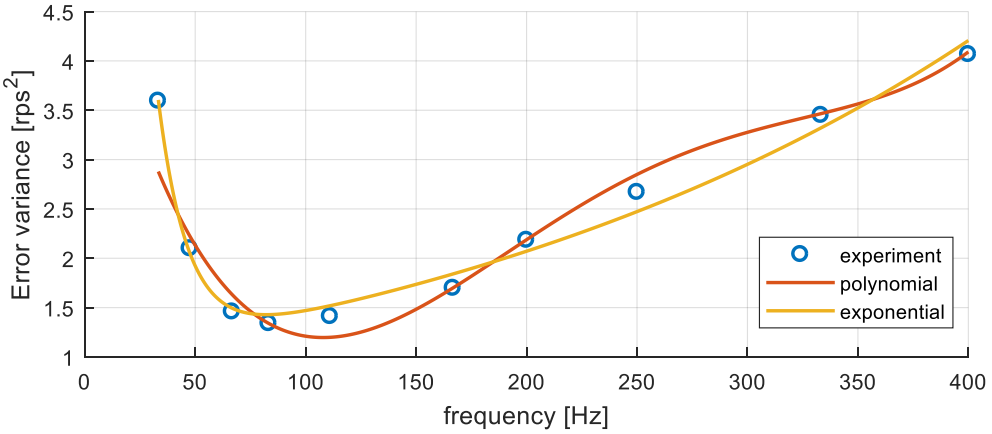

Figure 4-9. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±1.5-V disturbance at the actuator

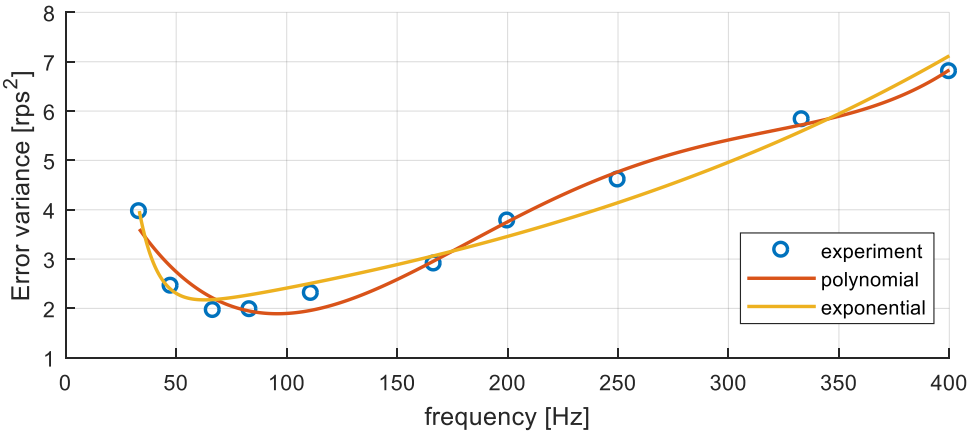

Figure 4-10. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±2-V disturbance at the actuator

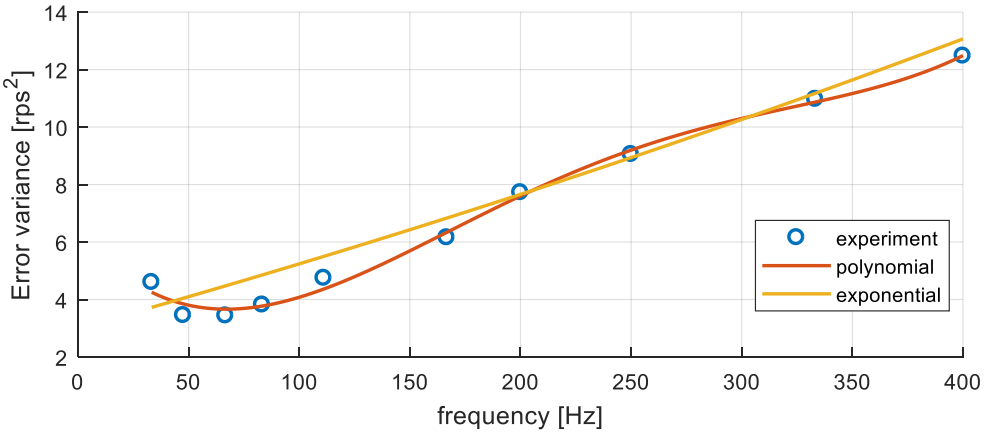

Figure 4-11. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±3-V disturbance at the actuator

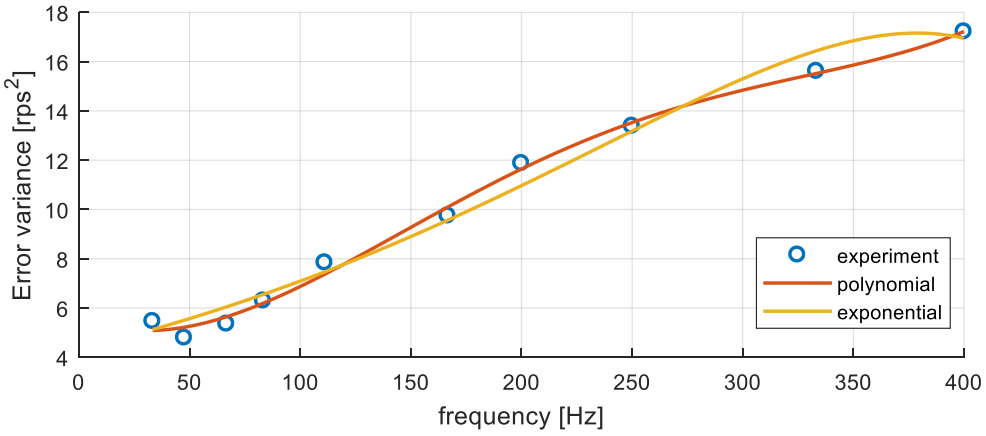

Figure 4-12. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±4-V disturbance at the actuator

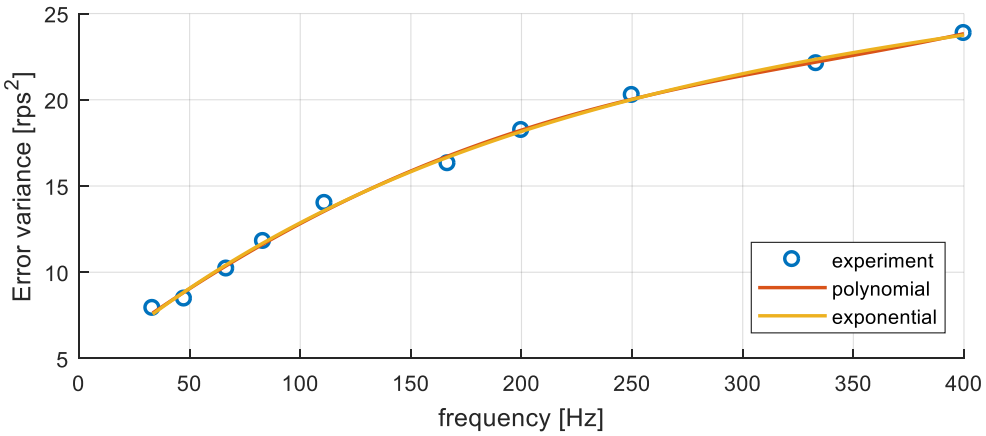

Figure 4-13. PIFs using 4th-order polynomial and exponential approximations of a DC motor system with a ±6-V disturbance at the actuator

Table 4.2. Coefficients of special form of 4th-order polynomial approximations (Eq. 4.3) for PIFs of the DC motor system with fixed standard deviations of disturbances

Disturbance	Coefficients			R-square
	p_1	p_2	p_3	
±0 (σ=0)	6.214×10^{-10}	272.9	0.0495	0.9357
±0.2 (σ=0.067)	6.932×10^{-10}	264.7	0.0941	0.9384
±0.4 (σ=0.133)	7.434×10^{-10}	259.9	0.1890	0.9365

Table 4.3. Coefficients of general form of 4th-order polynomial approximations (Eq. 4.2) for PIFs of the DC motor system with fixed standard deviations of disturbances

Disturbance	Coefficients					R-square
	a	*b*	*c*	*d*	*e*	
±0.8 (σ=0.267)	1.649×10^{-9}	-1.706×10^{-6}	6.205×10^{-4}	–0.0896	4.804	0.9525
±1 (σ=0.333)	1.677×10^{-9}	-1.739×10^{-6}	6.331×10^{-4}	–0.0896	4.865	0.9418
±1.5 (σ=0.5)	1.903×10^{-9}	-1.967×10^{-6}	7.1×10^{-4}	–0.0941	5.295	0.9298
±2 (σ=0.667)	2.793×10^{-9}	-2.8×10^{-6}	9.719×10^{-4}	–0.1189	6.590	0.9800
±3 (σ=1)	2.889×10^{-9}	-2.836×10^{-6}	9.38×10^{-4}	–0.0910	6.344	0.9942
±4 (σ=1.333)	2.423×10^{-9}	-2.298×10^{-6}	6.947×10^{-4}	–0.0362	5.606	0.9955
±6 (σ=2)	4.121×10^{-10}	-1.573×10^{-7}	-1.217×10^{-4}	0.0956	4.594	0.9975

Table 4.4. Coefficients of exponential approximations for PIFs of the DC motor system with fixed standard deviations of disturbances

Disturbance	Coefficients				R-square
	α	β	γ	δ	
±0 (σ=0)	21.62	–0.06241	0.9812	–0.01144	0.9991
±0.2 (σ=0.067)	16.92	–0.05206	0.317	–0.002884	0.9979
±0.4 (σ=0.133)	14.94	–0.04772	0.2049	0.001464	0.9983
±0.8 (σ=0.267)	17.33	–0.05322	0.3847	0.003039	0.9985
±1 (σ=0.333)	21.8	–0.05982	0.5332	0.003234	0.9976
±1.5 (σ=0.5)	29.6	–0.07474	1.0200	0.003542	0.9854
±2 (σ=0.667)	103.6	–0.1174	1.6810	0.003609	0.9730
±3 (σ=1)	4159	3.992×10^{-4}	–4156	3.943×10^{-4}	0.9665
±4 (σ=1.333)	–8657.5	0.007077	8661.813	0.007076	0.9841
±6 (σ=2)	23.04	2.834×10^{-4}	–18.87	–0.005547	0.9973

4.3.2 Performance index function for the DC motor system in terms of disturbance with fixed sampling frequencies

The PIF for the DC motor system with fixed sampling frequencies can be defined in terms of disturbance based on experimental data in Table 4.1. Unlike section 4.3.1, when the performance of the system with respect to various standard deviations of disturbances with fixed sampling frequencies is approximated, 2nd-order polynomials as shown in Eq. 4.4 are enough to describe the system performance. R-squared values were higher than 0.99 except for the case of the fixed 33.3 Hz sampling frequency with an R-squared value of 0.93. Figs. 4-14 to 4-23 present PIFs using 2nd-order polynomial approximations of the DC motor system with fixed sampling frequencies and various standard deviations of disturbances. To calculate the coefficients of each approximation, a linear least squares algorithm in MATLAB cftool was used. Approximation coefficients and R-squared values for each figure are listed in Table 4.5.

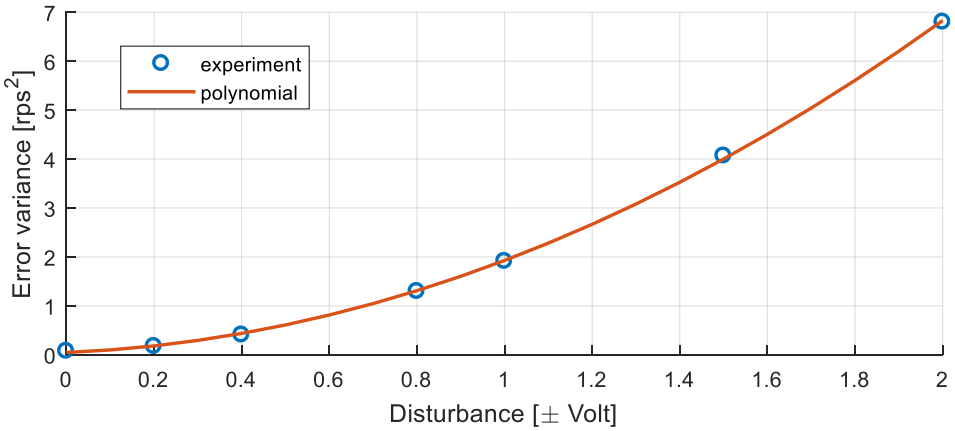

Figure 4-14. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 400 Hz

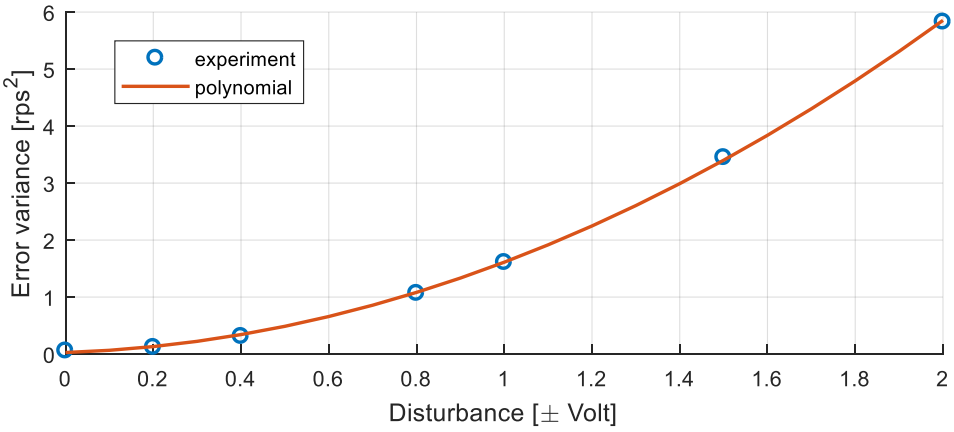

Figure 4-15. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 333 Hz

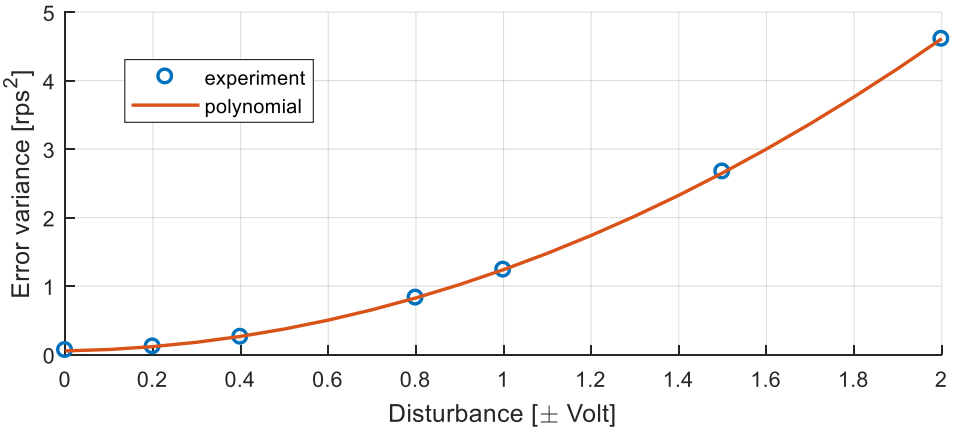

Figure 4-16. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 250 Hz

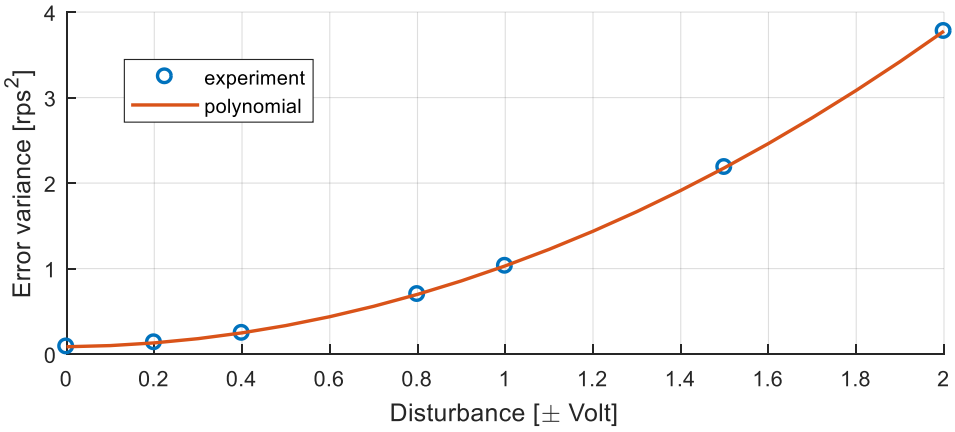

Figure 4-17. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 200 Hz

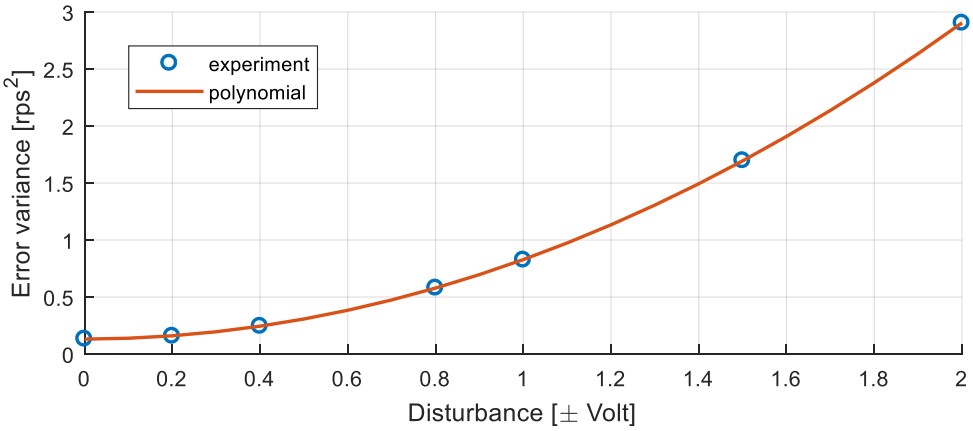

Figure 4-18. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 166.7 Hz

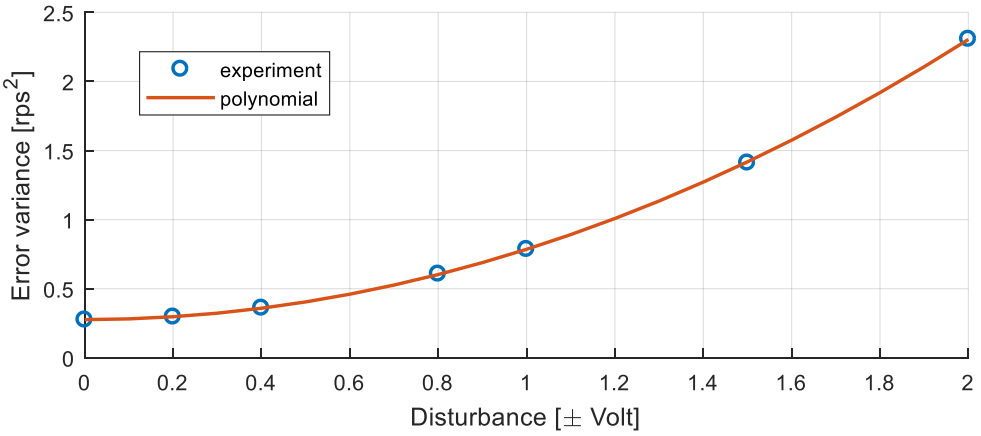

Figure 4-19. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 111.1 Hz

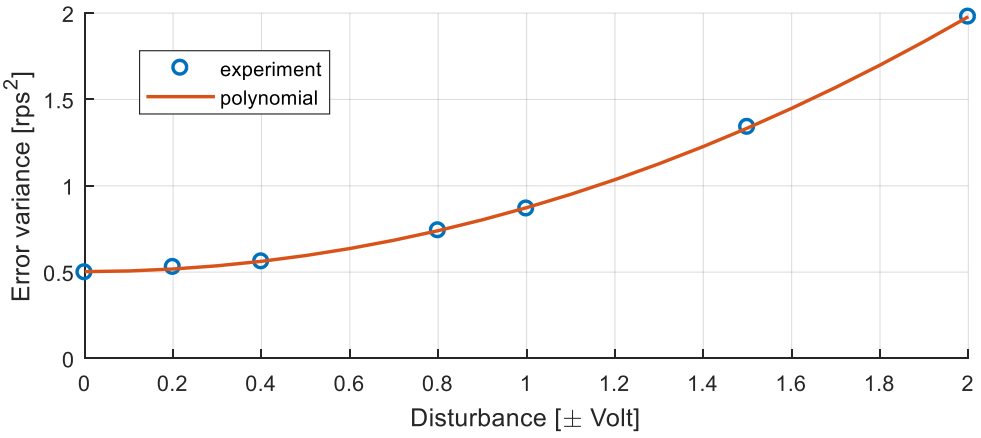

Figure 4-20. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 83.3 Hz

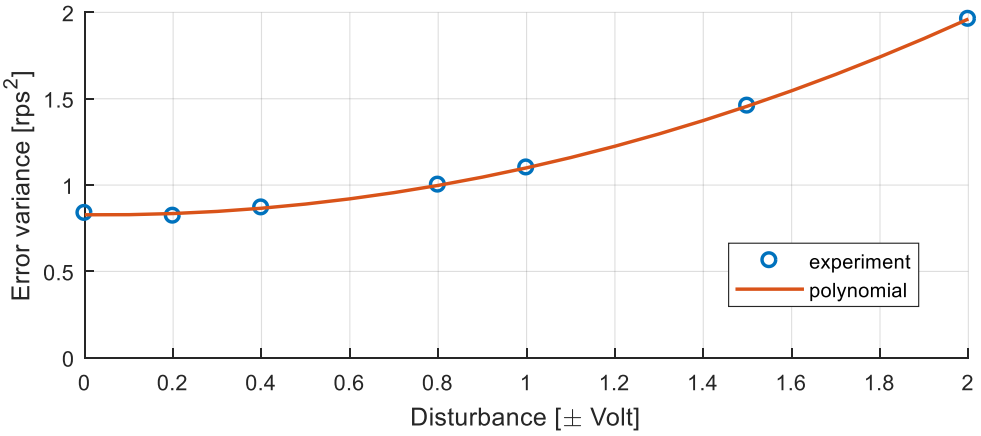

Figure 4-21. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 66.7 Hz

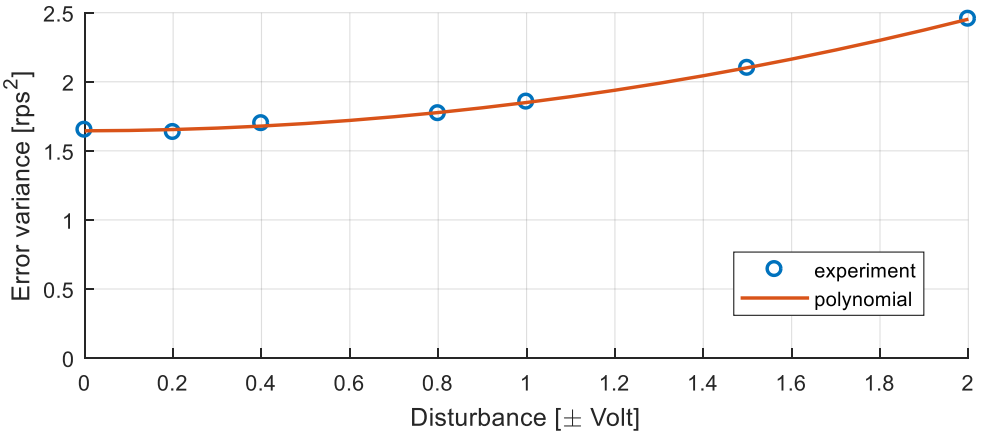

Figure 4-22. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 47.6 Hz

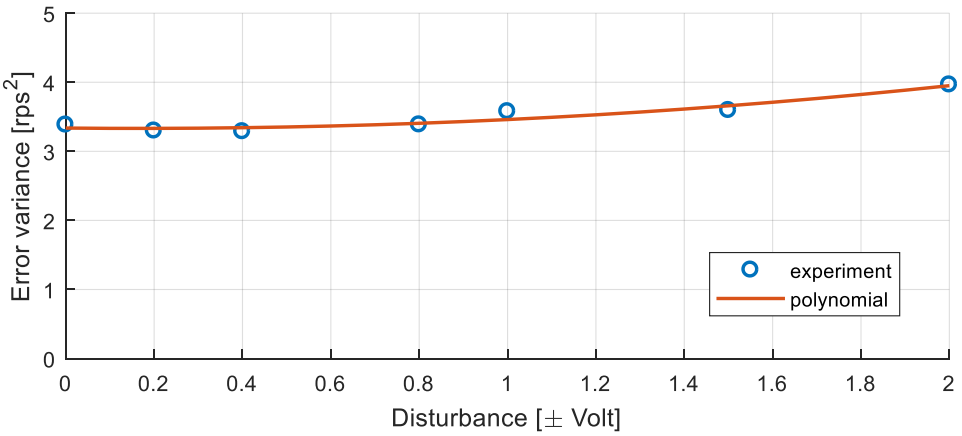

Figure 4-23. PIF using 2nd-order polynomial approximations of a DC motor system with various standard deviations of disturbances at the actuator for a fixed sampling frequency of 33.3 Hz

Table 4.5. Coefficients of 2nd-order polynomial approximations for PIFs of the DC motor system with a fixed sampling frequencies

Sampling frequency (Hz)	Coefficients			R-square
	a	*b*	*c*	
400	1.511	0.3697	0.04693	0.9998
333	1.329	0.2554	0.0274	0.9997
250	1.093	0.09234	0.05486	1
200	0.8998	0.04598	0.08578	1
166.7	0.6894	0.008149	0.131	1
111.1	0.5048	0.004275	0.276	1
83.3	0.3679	0.002292	0.5029	0.9999
66.7	0.2951	–0.02332	0.828	0.9997
47.6	0.1989	0.006186	1.645	0.9984
33.3	0.1838	–0.0607	3.337	0.9274

4.3.3 3-D PIF for the DC motor system

After accumulating the entire data and approximated PIFs, we can plot the 3-D PIF versus various sampling frequencies and maximum disturbances together as presented in Fig. 4-24.

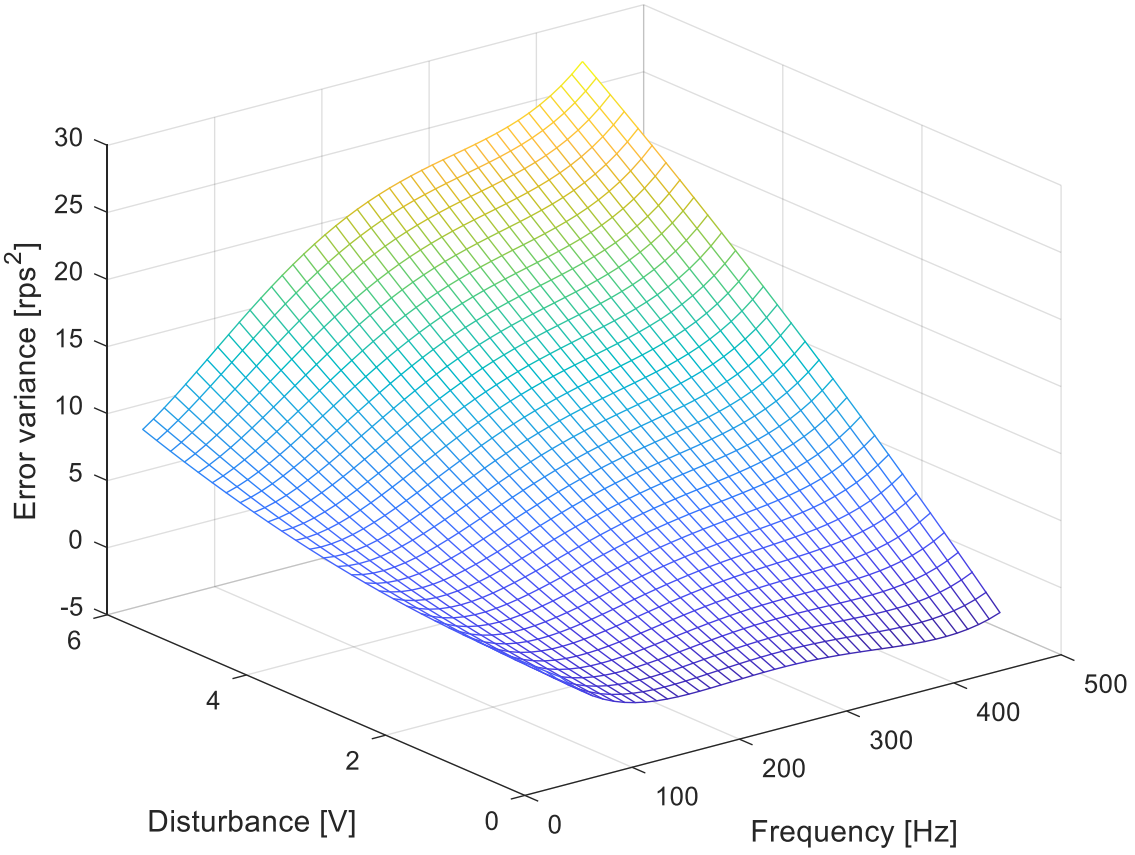

Figure 4-24. 3-D PIF versus various sampling frequencies and maximum disturbances

This proves that, unlike traditional continuous or discrete control systems, the faster sampling frequency does not always ensure less error variance and better system performance. A universal optimal sampling frequency does not exist. The error variance also changes when the sampling frequency or the maximum disturbance changes. The standard deviation of disturbance as well as the sampling frequency affects system performance. Thus, these two factors should be considered to find the optimal sampling frequency for the DC motor system. For Fig. 4-24, the 3-D PIF versus various sampling

frequencies and maximum disturbances can be approximated by the 6th-degree polynomial method as shown in Eq. 4.6.

$$\begin{aligned} J_i(f_i^k, d_i^k) = \sum_{k=1}^{M} (&6.841 - 0.1398\times(f_i^k) - 0.8867\times(d_i^k) + 9.265\times10^{-4}\times(f_i^k)^2 \\ &+0.02096\times(f_i^k)(d_i^k) + 0.1439\times(d_i^k)^2 - 2.552\times10^{-6}\times(f_i^k)^3 \\ &-5.094\times10^{-5}\times(f_i^k)^2(d_i^k) + 0.001213\times(f_i^k)(d_i^k)^2 + 2.453\times10^{-9} \\ &\times(f_i^k)^4 + 7.932\times10^{-8}\times(f_i^k)^3(d_i^k) - 4.061\times10^{-6}\times(f_i^k)^2(d_i^k)^2)dt \end{aligned}, \quad (4.6)$$

Figure 4-24 and Eq. 4.6 can be an important tool in calculating optimal bandwidth allocation. During the experiment, error variance can be calculated for every predefined loop and the sampling frequency is already known. Then the Eq. 4.6 becomes a 2nd-order polynomial equation in terms of the maximum disturbance. Thus, the maximum disturbance can be estimated by solving the 2nd-order equation with sampling frequency and error variance. After the maximum disturbance is estimated and is inserted into Eq. 4.6, the PIF becomes a 4th-order polynomial equation in terms of sampling frequency. Therefore, the optimal sampling frequency can be attained, which results in the least error variance from the 3-D PIF along with the estimated maximum disturbance.

4.4 Optimal Bandwidth Allocation Using the PIF Method

Figure 4.25 presents a flowchart of dynamic bandwidth optimization for the NCS with disturbance and noise. First, when the program runs, the client PCs calculate error variance for predefined loops. Since the sampling frequency is already known, the 6th-degree polynomial PIF becomes a 2nd-order polynomial as a function of the maximum

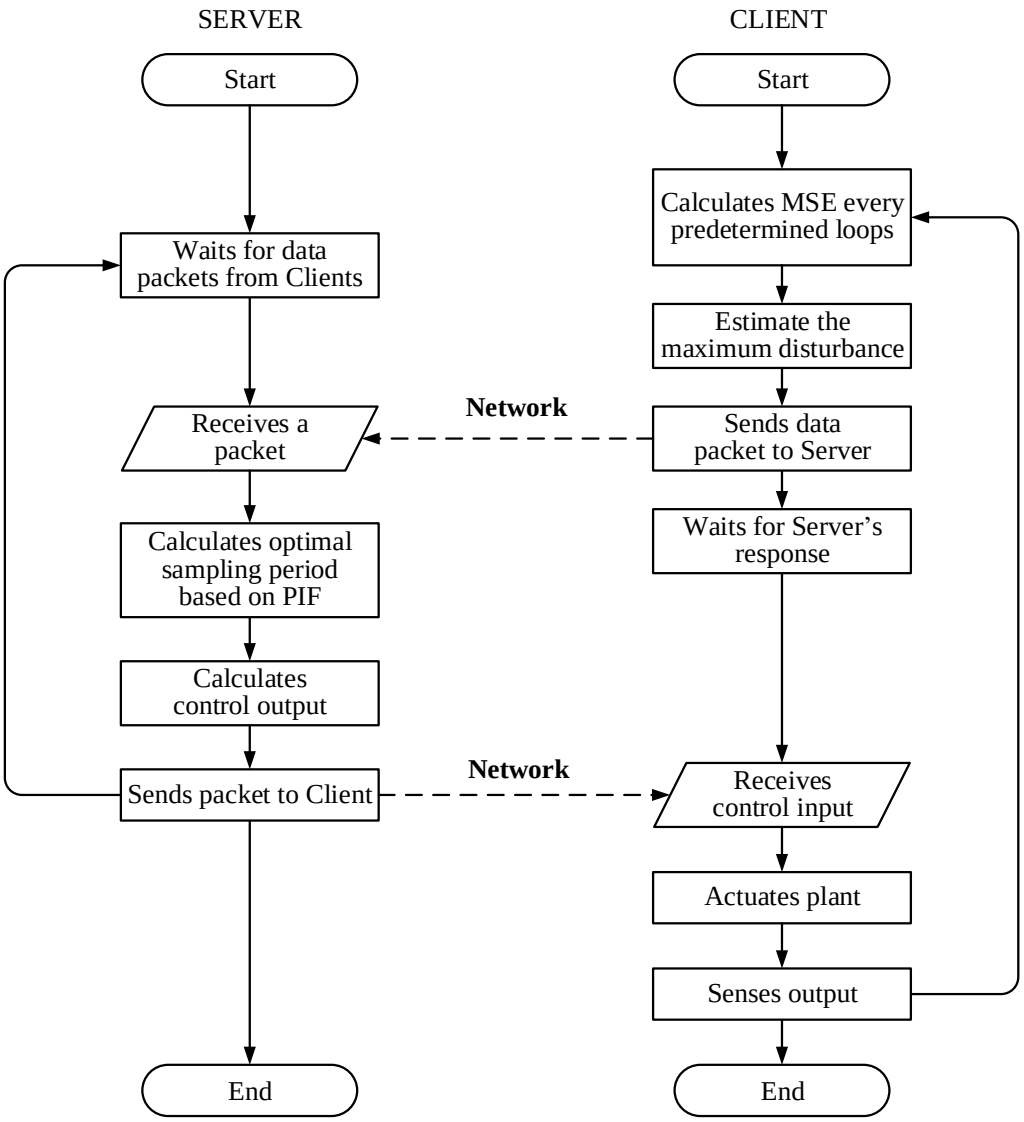

Figure 4-25. Flowchart for dynamic bandwidth optimization

disturbance. Then, the maximum disturbance can be estimated from the PIF by calculating the crossing point of the sampling frequency and error variance. Client PCs send each estimated disturbance and sensor data packet to the server. After the server receives a data packet, the 6^{th}-degree polynomial PIF becomes a 4^{th}-order polynomial as a function of

frequency and the optimal sampling frequency can be obtained from the PIF. After that, the server calculates the control input from the digital controller at the chosen optimal sampling frequency. In this research, a PI digital controller, which is converted using Tustin's method from continuous controller in Eq. 3.5, is used to control the DC motor system, since without integral control, the system output cannot reach its target value and a derivative controller is subject to high-frequency noises. Finally, a data packet to control the actuator is sent to each client PC. Once the client PC receives the data packet, it actuates the plant, senses the output of the plant, and resends the sensor data to the server. Both the server and client PCs run until the predefined control loops are excuted completely.

4.4.1 Disturbance estimation using the PIF method

Figures 4-26 to 4-28 present the experimental results that verify the PIF method. A white Gaussian disturbance was intentionally injected between the controller output and hardware. The standard deviation of the input disturbance, σ, was changed from 0.5 to 1 to 0.333 to 1, and to 0.667 for every 3000 loops as shown in Fig. 4-26. The standard deviations of disturbances vary from 0.333 to 1 because the dynamic bandwidth allocation method should find the optimal sampling frequency for each disturbance and it will obviously show the advantages of dynamic bandwidth allocation method compared to fixed sampling frequencies. The largest standard deviation of disturbance are chosen to 1, which corresponds to maximum 3-V disturbance because maximum output voltage for the DC motor system from the power supply is 5 V. The disturbance estimation results of the PIF are presented in Figs. 4-27 and 4-28. The PIF method estimated the original

disturbance with a small error and thus can be used to estimate the standard deviation of disturbance of the DC motor system in the NCS.

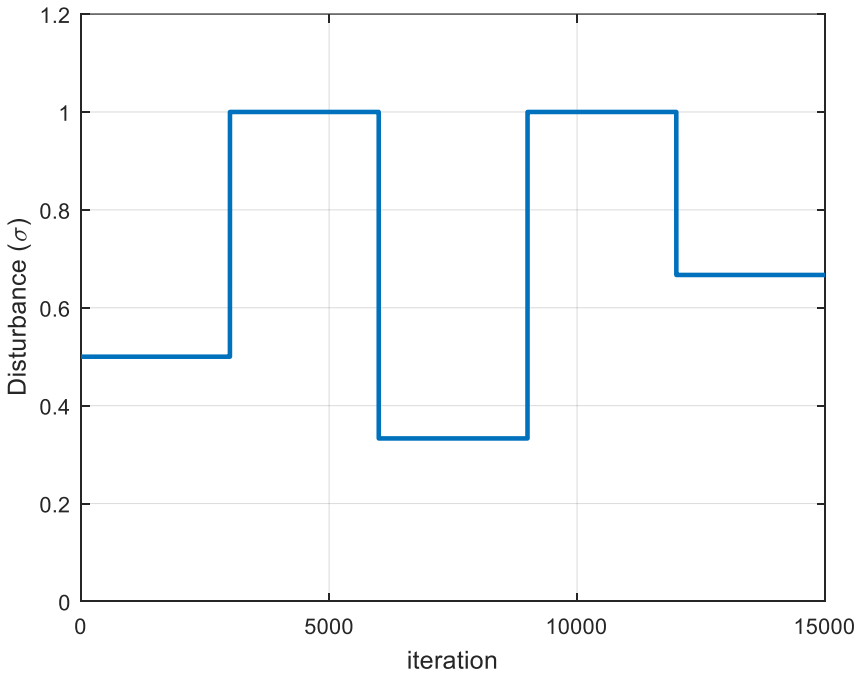

Figure 4-26. Original standard deviations of disturbances which were intentionally injected for the experiment

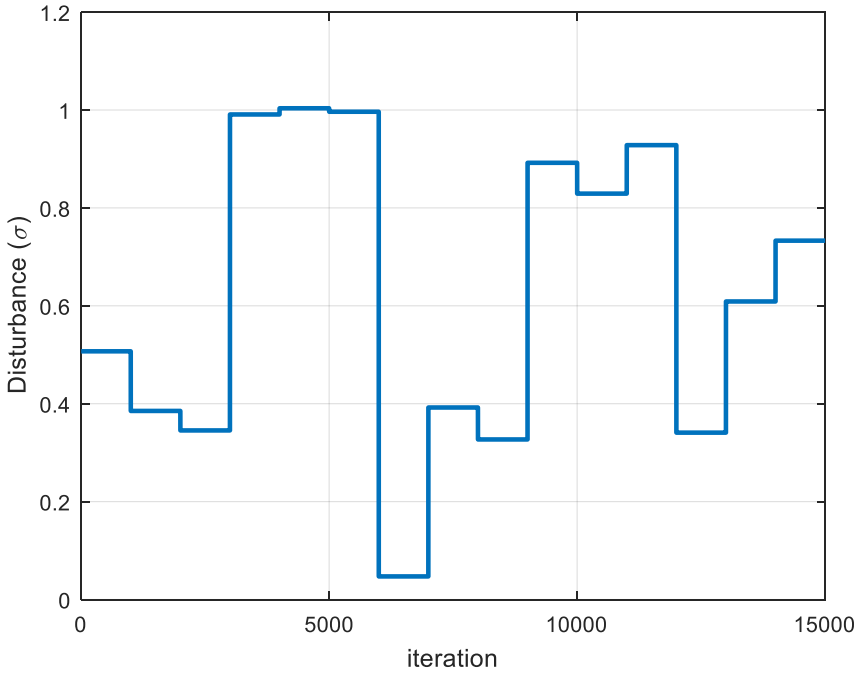

Figure 4-27. Estimated standard deviations of disturbances using the PIF method

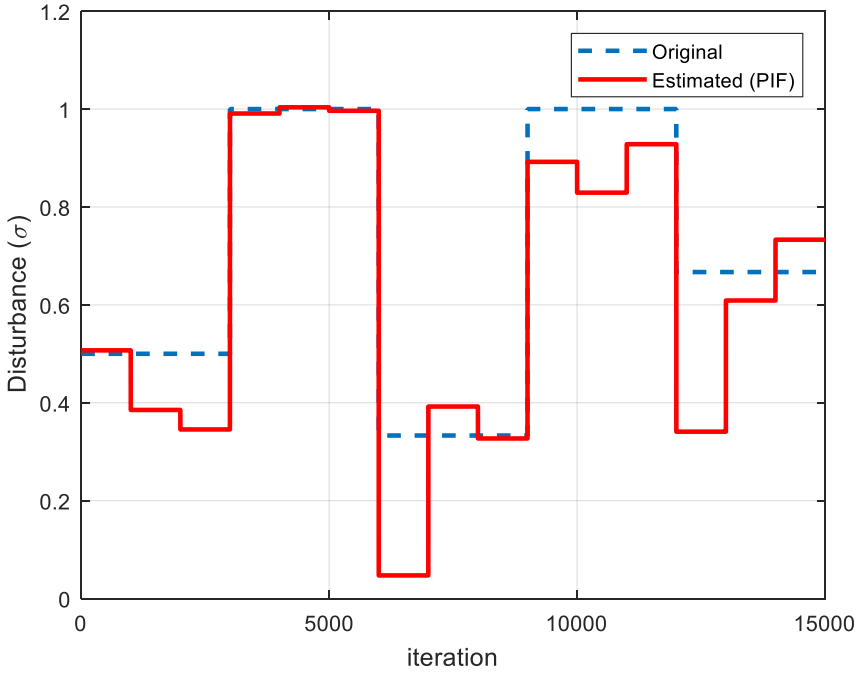

Figure 4-28. Comparison of standard deviations of disturbances between the original and the estimated using the PIF

4.4.2 Optimal bandwidth allocation for a DC motor system using the PIF

Figures 4-29 and 4-30 present sampling frequencies and bandwidth utilization changes based on the PIF when the amount of disturbance changed during 15 000 iterations as shown in Fig. 4-26. As explained at the beginning of this section, with error variance and sampling frequency information, standard deviations of disturbances can be estimated from the 6^{th}-order polynomial PIF of the Eq. 4.6 as shown in Fig. 4-27. If the estimated standard deviation of disturbance replaces actual disturbance in Eq. 4.6. The PIF becomes 4^{th}-order polynomial as a function of sampling frequency. Optimal sampling frequency can be calculated from the 4^{th}-order PIF as shown in Fig. 4-29. The average BU of the dynamic bandwidth allocation method using the PIF is about 13.15%. As shown in Eq. 3.6, BU is propotional to sampling frequency. Thus, changes of sampling frequency and BU look identical as shown in Fig. 4-29 and 4-30.

Figure 4-31 presents comparison of the IAEs between fixed sampling periods (3 ms, 6 ms, 9 ms, 12 ms, 15 ms, 21 ms, and 30 ms) and the dynamic bandwidth allocation method using the PIF. The dynamic bandwidth allocation method using the PIF shows the best performance at 16 615 IAE and uses an economical average BU of 13.15%.

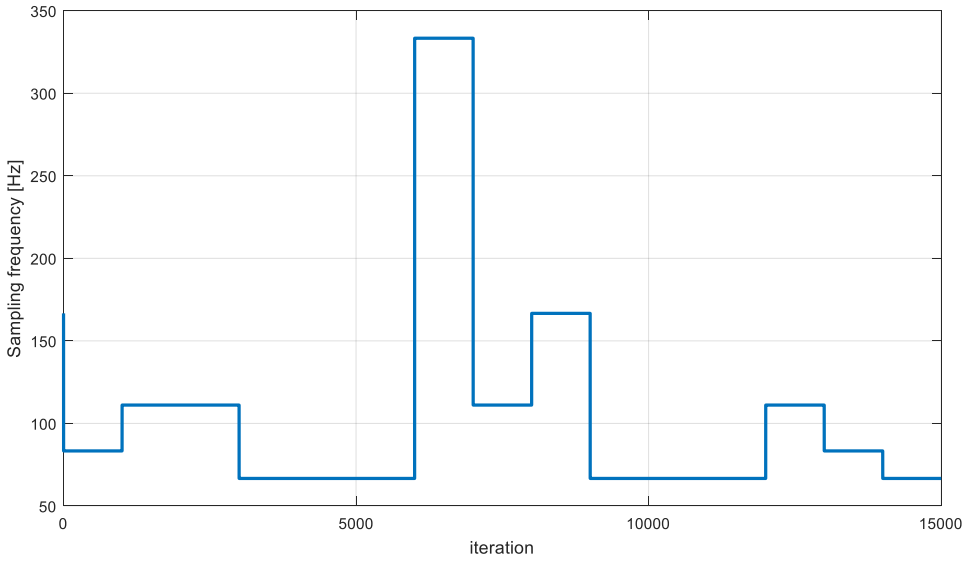

Figure 4-29. Changes of optimal sampling frequencies using the PIF during 15 000 iterations when various standard deviations of disturbances were injected into the system

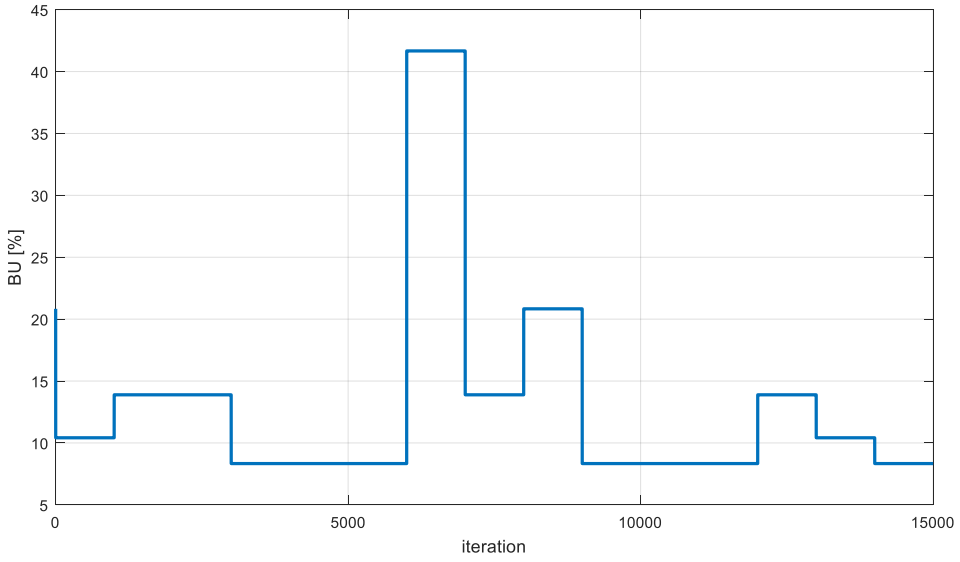

Figure 4-30. Optimal bandwidth utilization using the PIF during 15 000 iterations when various standard deviations of disturbances were injected into the system

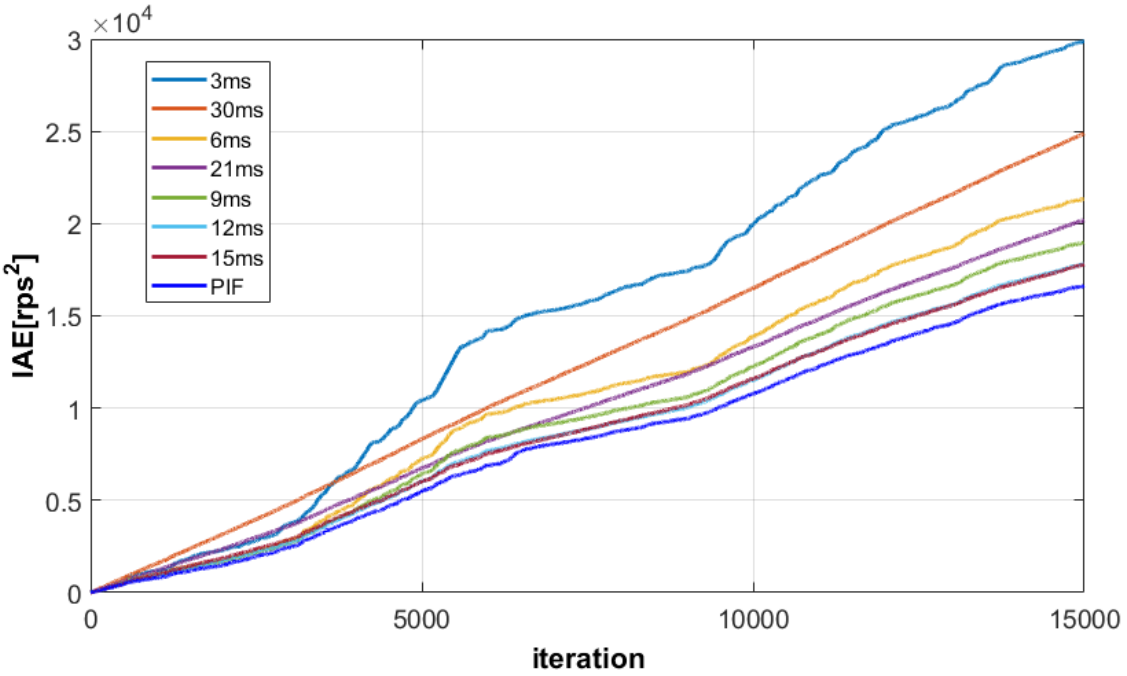

Figure 4-31. Comparison of the IAEs between fixed sampling periods (3 ms, 6 ms, 9 ms, 12 ms, 15 ms, 21 ms, and 30 ms) and the dynamic bandwidth allocation method using the PIF

4.4.3 Optimal bandwidth allocation for a multi-client system using the PIF

Equation 4.7 presents a constrained nonlinear optimization problem to find the optimal bandwidth allocation for a multi-client system using the PIF.

$$\begin{aligned} \min_{f\in\Omega} J &= \min_{f\in\Omega} \sum_{i=1}^{N} w_i^k P_i \\ &= \min_{f\in\Omega} \sum_{i=1\ i}^{N} w_i^k (a_i (f_i^{\,k})^4 + b_i (f_i^{\,k})^3 + c_i (f_i^{\,k})^2 + d_i f_i^{\,k} + e_i) \end{aligned} \tag{4.7}$$

Subject to

$$\sum_{i=1}^{N} \tau_i^k f_i^{\,k} \le B, \qquad \forall_k = 1,\cdots,M, \quad \forall_i = 1,\cdots,N$$

$$33 \text{ Hz} \le f_i^{\,k} \le 450 \text{ Hz}$$

,

where J is the cost function of the total system, which is the summation of multiplication of weights and performance of each system, P_i is the performance index of system i, N is the total number of systems, w_i^k is the weight of system i at iteration k, which can be selected by engineer to put more available bandwidth to a certain system, f_i^k is the sampling frequency of system i at iteration k, τ_i^k is the total time delay of system i at iteration k, B is the available bandwidth utilization, a_i, b_i, c_i, d_i, and e_i are coefficients. From the previous experiment as shown in Fig. 4-29, the range of sampling frequency was chosen from 33 Hz to 450 Hz because high sampling frequency such as 450 Hz are subject to large standard deviation of disturbance and low sampling frequency such as 33 Hz presents large error variance despite of small standard deviation of disturbance. To determine the optimal sampling frequency for each client in the NCS with disturbance and noise, sequential quadratic programing (SQP) approach is used. SQP is one of the most effective methods for nonlinearly constrained optimization and generates steps by solving quadratic sub-problems [68]. SQP may only give a local minimum. The PIF of the DC motor system, when estimated disturbance is provided, is a 4^{th}-order polynomial that has two local minima. Thus, a global minimum can be obtained by (1) choosing the initial point of iteration as a lower boundary of the sampling frequency, (2) calculating the local minimum of the cost function, (3) choosing the other initial point of iteration as an upper boundary of the sampling frequency, (4) calculating the local minimum of the cost function, (5) comparing two local minimum, and (6) selecting optimal sampling frequency for each system with a lower local minimum, which is the

global minimum. In this research, since four DC motor systems was studied and the PIF of each DC motor system had two local minima, a total of 16 initial points were selected at the boundaries.

4.4.3.1 *Experimental result – Case 1: a* ±0.8-V *disturbance,* 10% *safety margin of BU,* 1.25-ms *total time delay, and the weights of DC motor systems:* 4, 3, 2, *and* 1, *repectively*

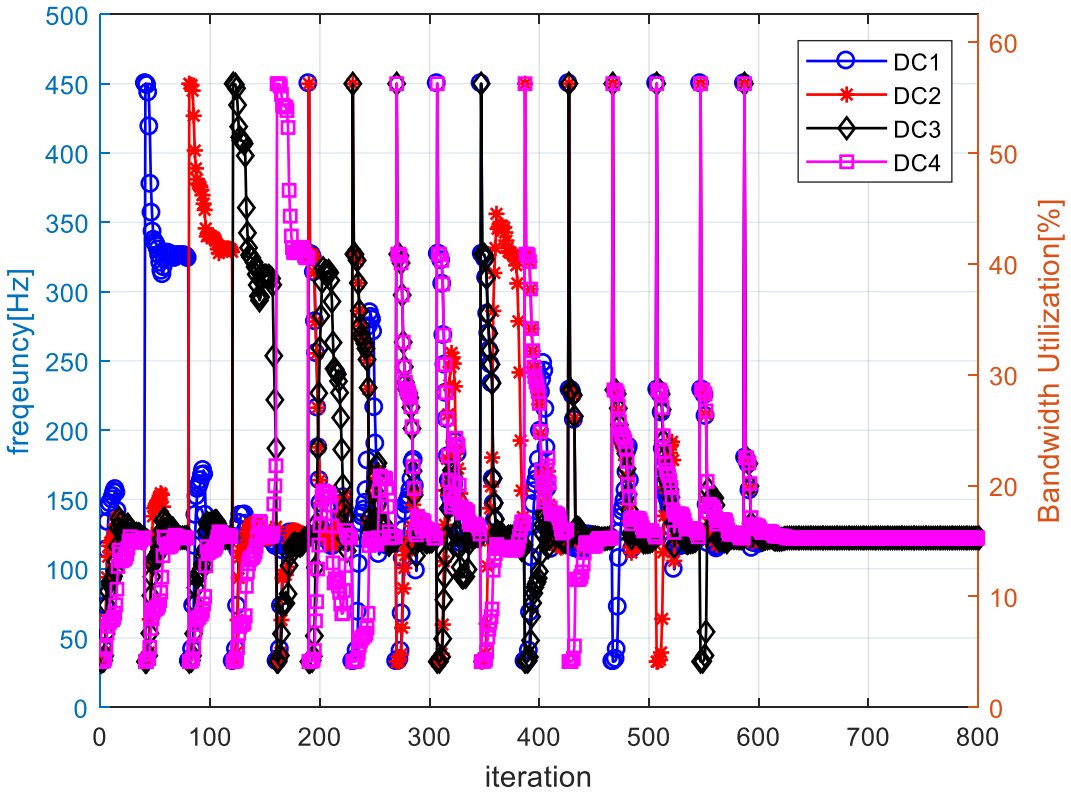

Figure 4-32. Profile of the sampling frequencies and BU changes for each DC motor during the experiment in Case 1

Figures 4-32 to 4-35 presents experimental results of case 1, where a ±0.8-V disturbance was intentionally injected, the safety margin of BU was 10%, there was a

1.25-ms total time delay, and the weight of each DC motor was 4, 3, 2, and 1, respectively. In this dissertation, the safety margin of BU was chosen as 10%. Each DC motor system has average 1.25-ms total time delay and standard deviations of round-trip time between server 1 to client 2 is 0.012 and between client 2 to server 1 for DC motor system are 0.012 and 0.027, respectively, as shown in the section 3.6. Network induced time delay is stochastic and if it is assumed to show the normal distribution, 99.99% of the data are within 4 standard deviations of the mean. Thus, maximum time delay for the DC motor system can be calculated by $\{(0.397+0.027\times4) + (0.378+0.012\times4)\}/2 + 0.866 = 1.332$ ms in the worst case. Using Eq. 3.6 and due to the fact that total summation of BU should not be greater than 1, maximum summation of sampling frequencies of four DC motors can be calculated by $\sum f_i^k = 1/\tau_i^k = 750.75$ Hz in the worst case. When summation of sampling frequencies of four DC motors are 750.75 Hz, total BU with average time delay is calculated by $1.25\times10^{-3}\times750.75\times100 = 93.84\%$. Thus, at least 6.16% of safety margin of BU is required to perform optimal bandwith allocation for case 1. When a maglev and a wheelchair robot system are included in the NCS as shown in section 4.4.3.2, additional 2.33% and 1.23% of safety margin of BU are calculated using the same method as the DC motor system used. Thus, 10% of safety margin of BU is selected to validate the optimal bandwidth allocation for a multi-clinet and multi-server system in this dissertation.

Figure 4-32 presents how optimal sampling frequency is calculated when the iterations increase, and Fig. 4-33 is a reduced version of the profile of the sampling frequency and BU changes for each DC motor during the experiment in Case 1 to see the details of iterations for finding one local minimum. As shown in Fig. 4-33, the initial

starting points of all DC motor systems was 33 Hz, and optimal sampling frequency was calculated by solving the nonlinear constraint problem in Eq. 4.7 at f_1 = 122.2 Hz, f_2 = 122.2 Hz, f_3 = 122.2 Hz and f_4 = 122.2 Hz, respectively, for 40 iterations. It took 626 iterations to complete the whole optimization. Several peaks are observed during iterations because 4th-order polynomials generally have two local minima and starting points of each DC motor are one of the boundaries. When one of initial point of DC motor systems is 450 Hz, peak is observed. 16 initial points at the boundaries are chosen and all local minima are compared to find the global minimum.

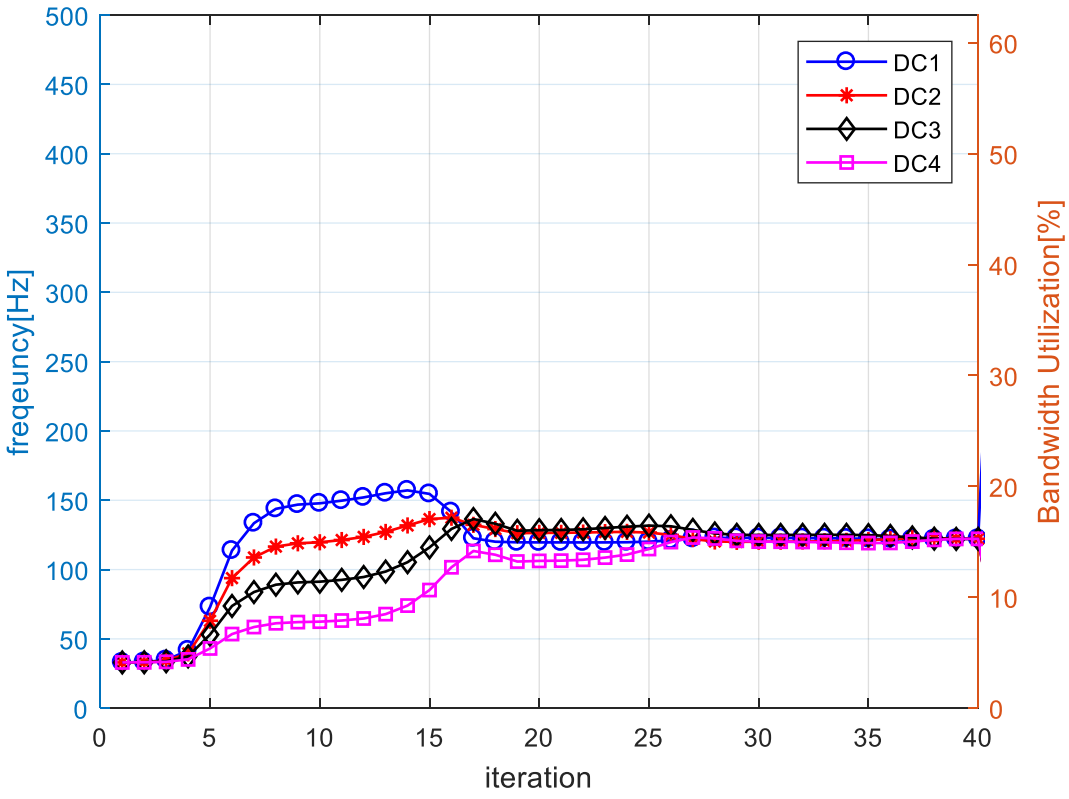

Figure 4-33. Reduced profile of the sampling frequencies and BU changes for each DC motor during the experiment in Case 1

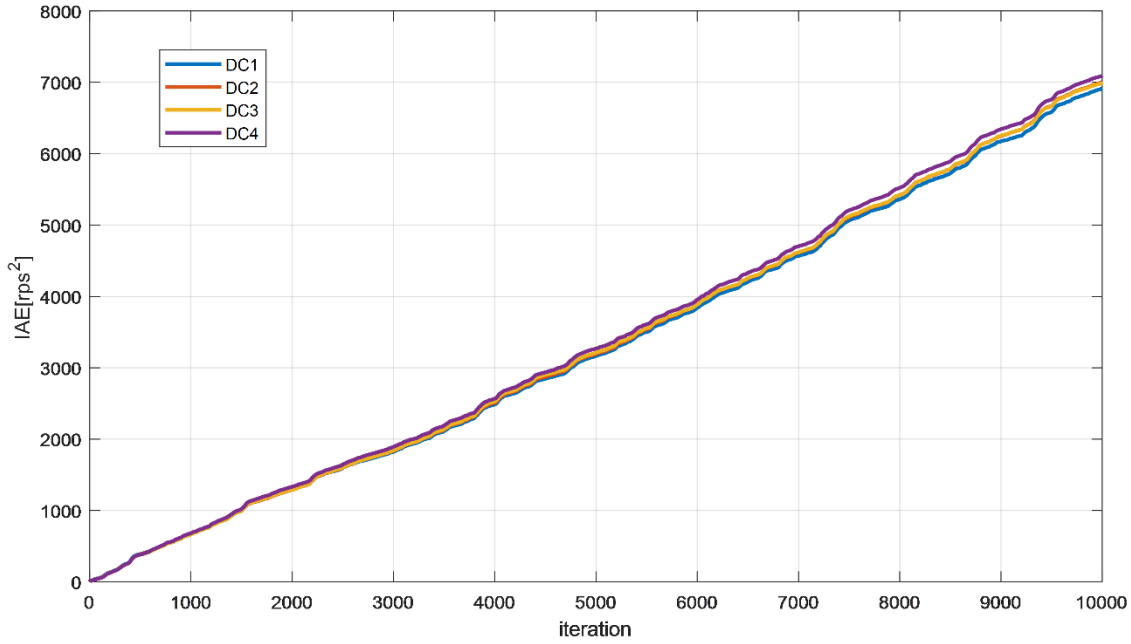

Figure 4-34. IAEs of the DC motors in the case of a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, and the weights of DC motor systems: 4, 3, 2, and 1, repectively

Figure 4-34 presents the experimental results of the PIF method in case 1. The optimal sampling frequencies of each system were 122.2 Hz, 122.2 Hz, 122.2 Hz and 122.2 Hz and the IAE values are 6865.8, 7061.6, 6917.1 and 7108.7, respectively. Although all four DC motors are assumed to have identical properties and specifications, actually motor specs are different, each client PC also exhibits different performance, and there could be unexpected time delays or packet dropouts. That might lead slight different values of IAEs. It is also possible that unexpected sensor noise arose for conducting experiments. Four DC motor systems require a total of 61.1% bandwidth to operate in an optimal sampling frequency for each system. Thus, a total of 90% bandwidth is enough to

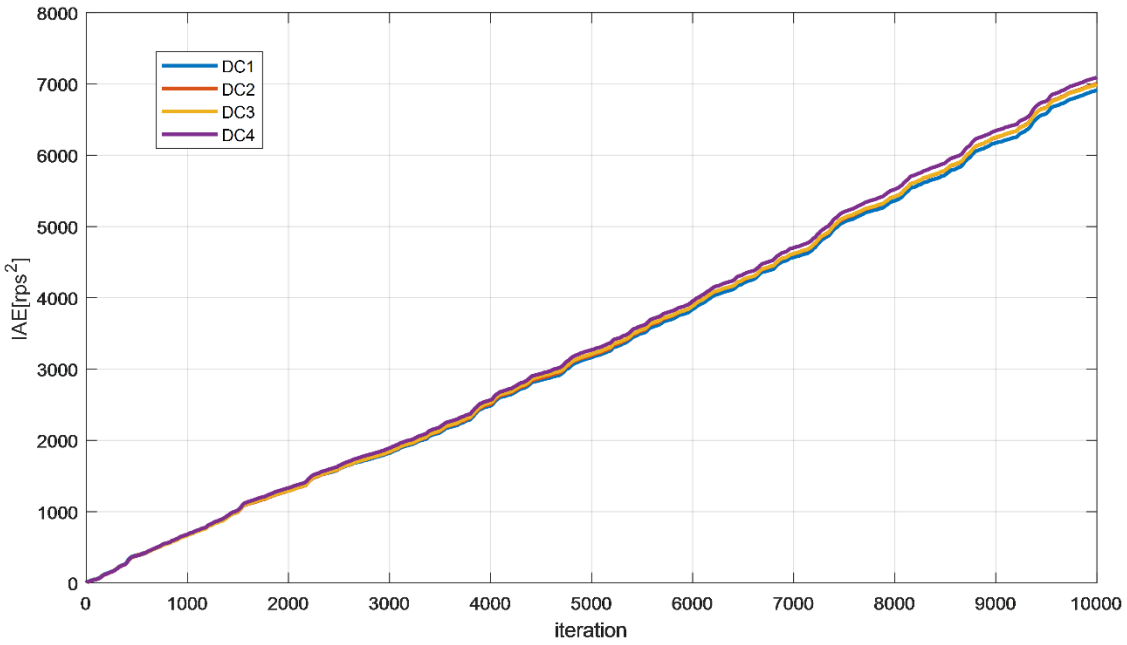

Figure 4-35. IAEs of the DC motors in the case of a ±0.8-V disturbance, 10% safety margin of BU, a 1.25-ms total time delay when the maximum bandwidth is utilized

operate in an optimal sampling frequency for four DC motor systems and the optimal sampling frequencies were identical in spite of the different weights of each system.

Figure 4-35 presents the experimental results of the PIF method where a disturbance was intentionally injected, the safety margin was 10%, the total time delay was 1.25-ms, and the weight of each DC motor was 4, 3, 2, and 1, respectively, with a maximum BU to compare the results with the optimal bandwidth allocation method. Since the safety margin was 10 %, sampling frequency of each system was 180 Hz and the IAEs were 6915.1, 7006.2, 6996.7 and 7094.7. Although a maximum BU is used and faster sampling frequencies were allocated to each system than frequencies using the optimal bandwidth allocation method, the optimal bandwidth allocation method using the PIF

presents a similar performance with a lower BU. Thus, the optimal bandwidth allocation method using the PIF are more economical.

4.4.3.2 *Experimental result – Case 2: a ±0.8-Vdisturbance, 10% safety margin of BU, 1.25-ms total time delay, and the weights of DC motors: 4, 3, 2, and 1, respectively, with a maglev and a wheelchair robot system*

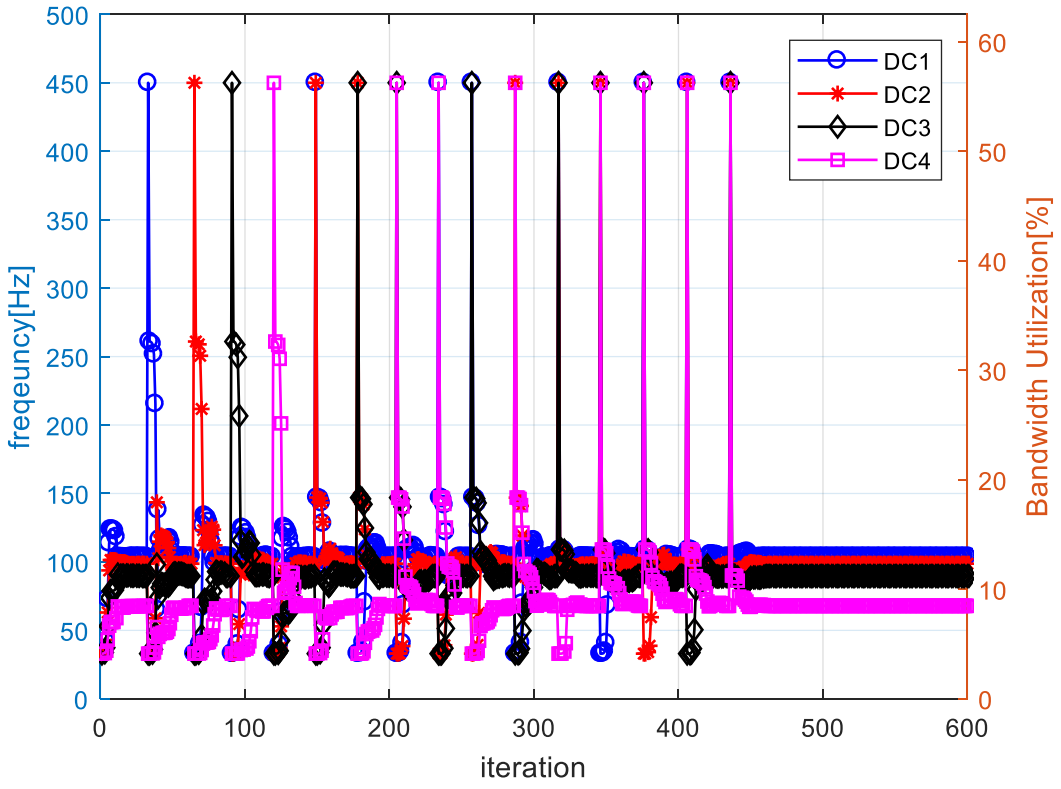

Figure 4-36. Profile of the sampling frequencies and BU changes for each DC motor during the experiment in Case 2

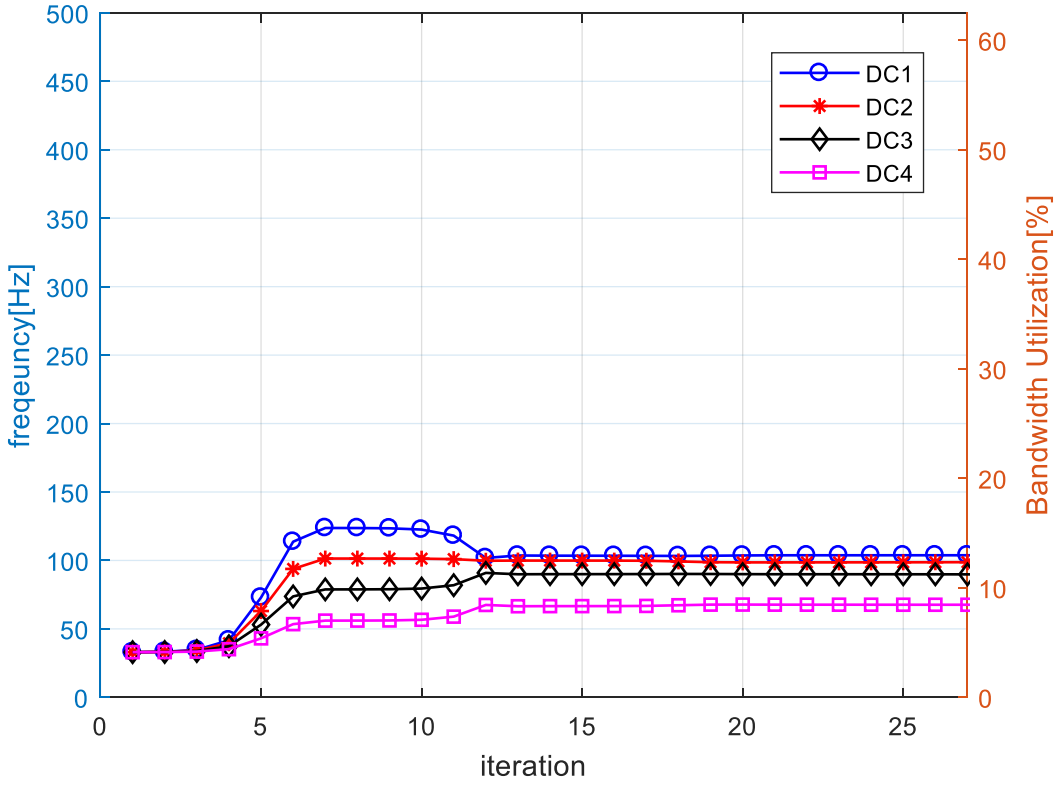

Figure 4-37. Reduced profile of the sampling frequencies and BU changes for each DC motor during the experiment in Case 2

The Maglev system uses 44% BU as Table 3.2 in 3-ms sampling time and the wheelchair robot uses 1% BU in 300-ms sampling time. Safety margin to avoid time delays or packet dropouts are 10%. Thus, four DC motors can utilize up to 45% BU.

Figure 4-36 shows the change in sampling frequency for the DC motor system during iterations to find the optimal bandwidth allocation. Fig. 4-37 is a reduced version of the profile of the sampling frequencies and BU changes for each DC motor during the experiment in Case 2. SQP algorithm with a 3-D PIF are used to determine optimal frequencies for each DC motor. As shown in Fig. 4-37, the initial starting point of each

DC motor system was 33 Hz and the optimal sampling frequency was calculated at f_1 = 103.73 Hz, f_2 = 98.69 Hz, f_3 = 89.66 Hz and f_4 = 67.91 Hz, respectively, for 33 iterations. It took 463 iterations to complete the whole optimization.

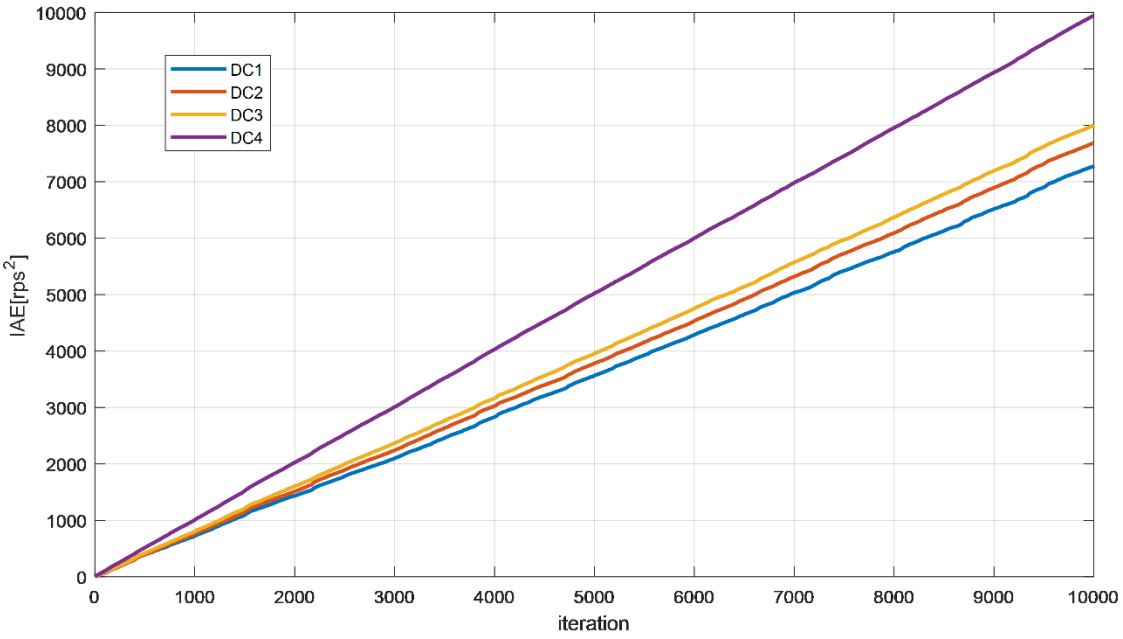

Figure 4-38. IAEs of the DC motors in the case of a ±0.8-V disturbance, a 10% safety margin of BU, 1.25-ms total time delay, and the weights of DC motors: 4, 3, 2, and 1, respectively, with a maglev and a wheelchair robot system

Figure 4-38 presents the experimental results of a PIF method where a ±0.8-V disturbance was intentionally injected, the safety margin of BU was 10%, total time delay was 1.25 ms, and the weight of each DC motor was 4, 3, 2, 1, respectively, with a maglev and wheelchair robot system. Since the maglev system with a 3 ms sampling time requires a 44% BU and the wheelchair robot uses 1% BU, only 45% of the bandwidth can be utilized. Optimal sampling frequencies of each system were 103.73 Hz, 98.69 Hz, 89.66 Hz and 67.91 Hz and the IAE values were 7281.7, 7689.2, 8002.9 and 9945.8. Due to the

different weights of each system, they had a different optimal sampling frequencies and system performance increased in the order of higher weight number.

Figure 4-39 presents the experimental results of the PIF method where a disturbance was intentionally injected, the safety margin was 10 %, and the total time delay was 1.25 ms with a maglev and wheelchair robot system with a maximum BU to compare results with the optimal bandwidth allocation method. Since 45 % of the bandwidth can be utilized, the sampling frequency of each system was 90 Hz and the IAE values were 7904.1, 8405.1, 8049.4 and 8436.6. Since an identical maximum sampling frequency was allocated to each system, all systems had similar IAEs. On the other hand, the optimal bandwidth allocation method with the weights of 4, 3, 2 and 1 presents a better performance which has larger number of weight.

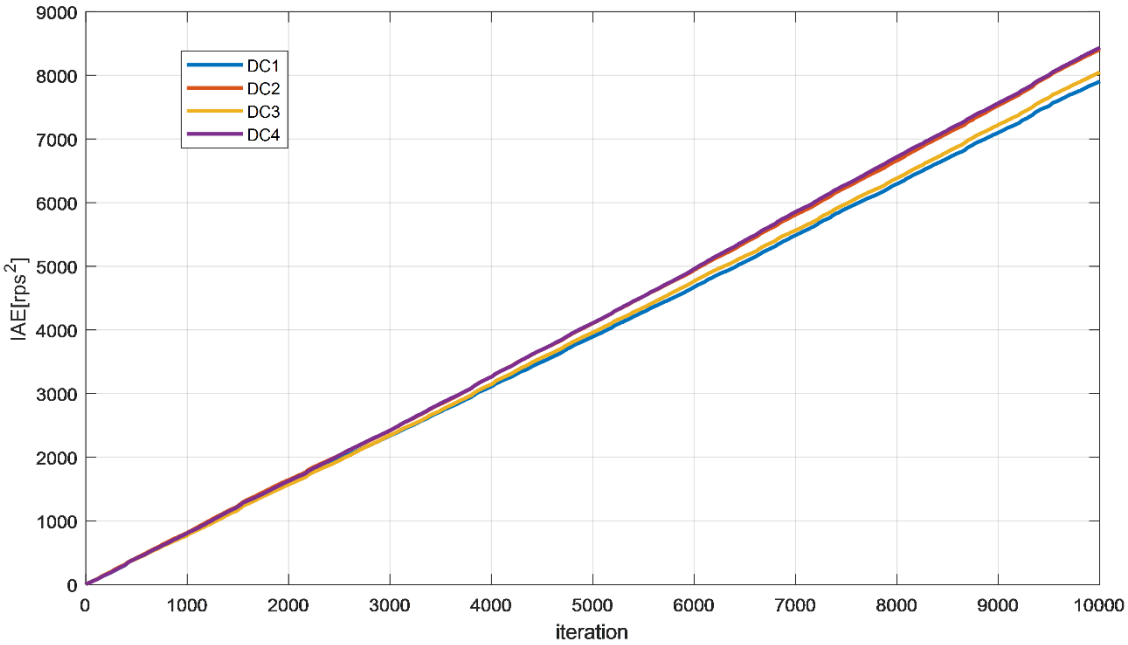

Figure 4-39. IAEs of the DC motors in the case of a ±0.8-V disturbance, 10% safety margin of BU, a 1.25-ms total time delay with a maglev and a wheelchair robot system when the maximum bandwidth is utilized

4.5 Conclusion of PIF Method for NCS with Disturbance and Noise to Find Optimal Bandwidth Allocation

In this section, the PIF method for NCS with disturbances and noises was used to find the optimal bandwidth allocation. The relationship between the sampling frequency and the performance of a system can be approximated by using a 4^{th}-order polynomial or exponential approximation. However, the exponential approximation approach requires more calculation time in real-time. Thus, a 4^{th}-order polynomial approximation was chosen in this research to determine the optimal sampling frequency. The relationship between the amount of disturbance and the performance of a system can be easily approximated by using a 2^{nd}-order polynomial. Thus, the relationship between the system performance and sampling frequency and the amount of disturbance can be approximated using 6^{th}-degree polynomials with two different variables such as sampling frequency and disturbance. When the error variance was calculated and the sampling frequency was informed, the standard deviations of disturbance was estimated using the PIF, which presents the relationship between disturbance and system performance with a fixed sampling frequency. After obtaining the amount of estimated disturbance, the safety margin of BU, the weight of each DC motor system and the total time delay, the optimal sampling frequency can be determined by solving the nonlinear constraint optimization method using SQP methods. Experimental results proved the effectiveness of the PIF method.

5. ARTIFICIAL NEURAL NETWORK METHOD FOR THE NCS WITH DISTURBANCE AND NOISE

5.1 Disturbance Estimation Using the ANN Method

Figure 5-1 shows the general diagram of an ANN which will be used to infer the relationship between the sampling frequency and the system performance for the NCSs with uncertainty. The inputs for the ANN are sampling period and the MSE values of the system. Outputs are the standard deviation of disturbance. In this dissertation, the ANN algorithm is used twice to calculate estimated standard deviation of disturbance and optimal sampling frequency.

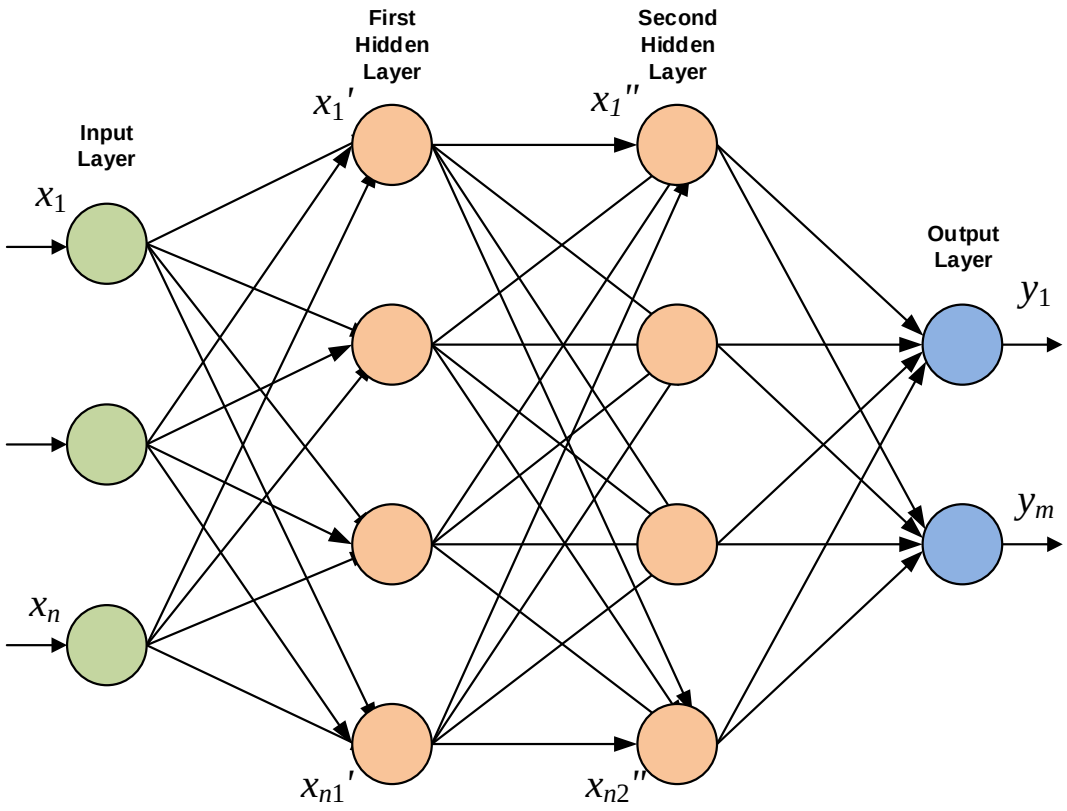

Figure 5-1. General diagram of an ANN

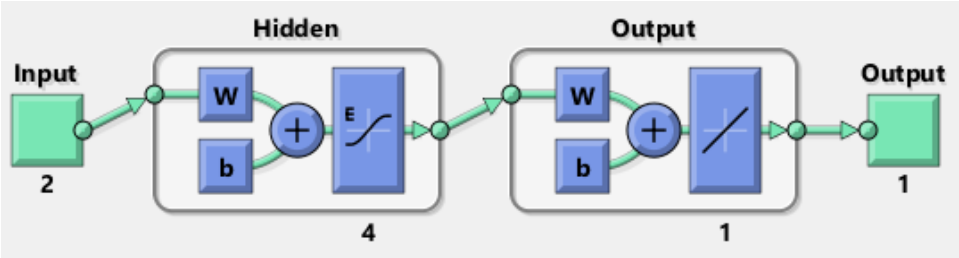

Figure 5-2. Diagram of an ANN for disturbance estimation of a DC motor system

A specific diagram of an ANN is presented in Fig. 5-2 to estimate the disturbance of a DC motor system. The two inputs are the sampling period and MSE, and the output is the estimated disturbance. To model an ANN for disturbance estimation for a DC motor system, MATLAB nftool is used and activation function and number of neurons and layers are found by trial and error. Four neurons and two layers are used for the ANN. The elliot-sigmoid function is used as an active function as follows:

$$\sigma(x) = \frac{1}{1+|x|} \cdot \tag{5.1}$$

Unlike other sigmoid transfer functions, it does not use exponential or hyperbolic functions and thus, reduces calculation time, which is essential to operate NCS in real-time. Data shown in Table 4.1 are used to train and validate the ANN.

Figures 5-3 to 5-6 present the training results of ANN to estimate the disturbance of a DC motor system. Most of errors are in ±0.2 V in Fig. 5-3, the R-squared value is 0.99 in Fig. 5-4, and the training performance is presented in Fig. 5-5. The trained data can follow target data with small errors in Fig. 5-6. Therefore, the ANN effectively found the relationship between inputs (sampling time and MSE values) and outputs (estimated disturbance). The ANN method can be used to estimate the disturbance of a DC motor

system. The bias of 1st layer are 0.473, 1.912, -2.744, and -3.941 and the weights of 1st layer are -0.769, -0.691, -3.771, -0.244, -2.337, 0.478, 1.486, and -4.708, repectively. The bias of 2nd layer is -0.714 and the weights of 2nd layer are -7.706, 3.260, -2.352, and -2.059, repectively.

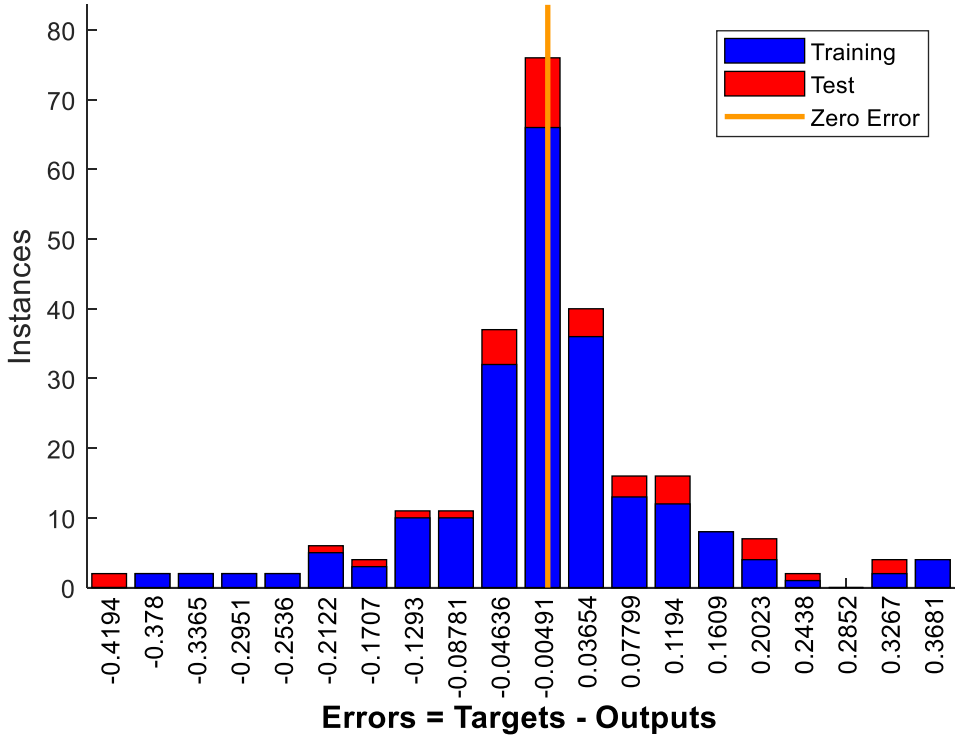

Figure 5-3. Error histogram of an ANN for disturbance estimation

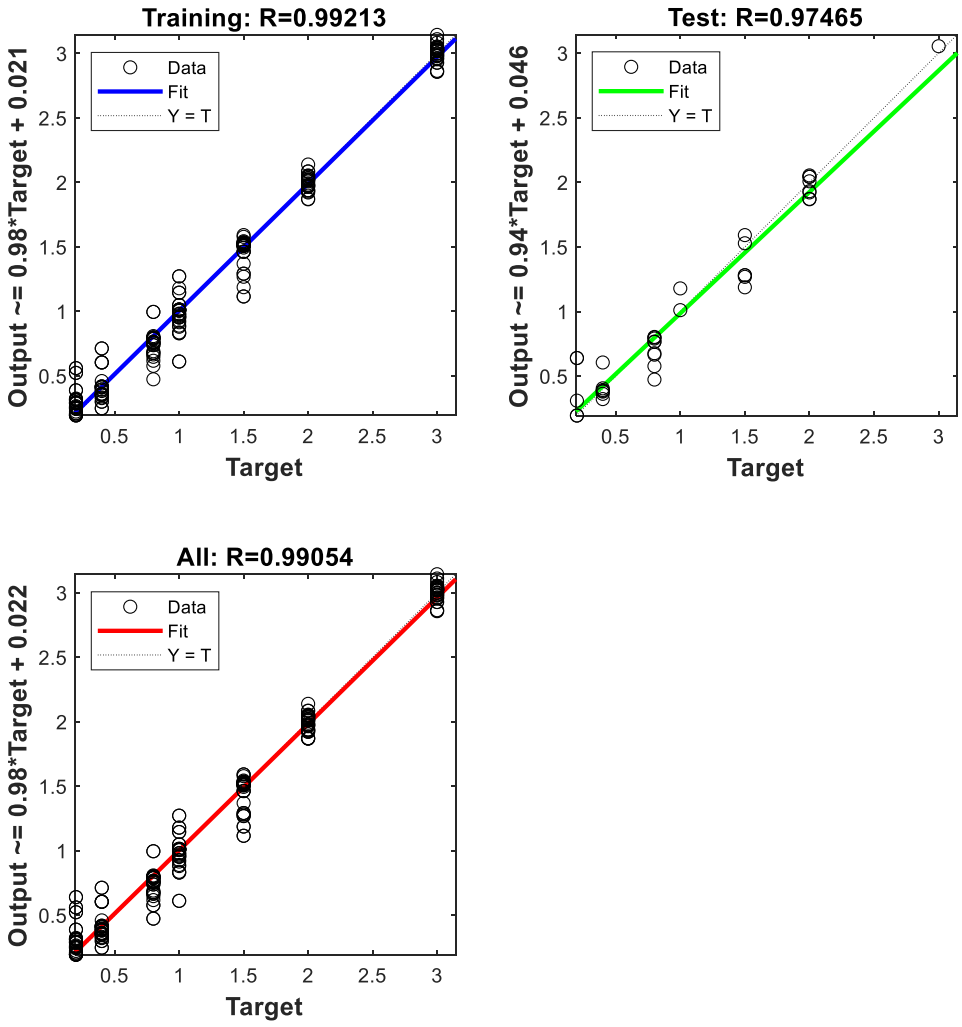

Figure 5-4. R-squared value of the ANN to estimate disturbance

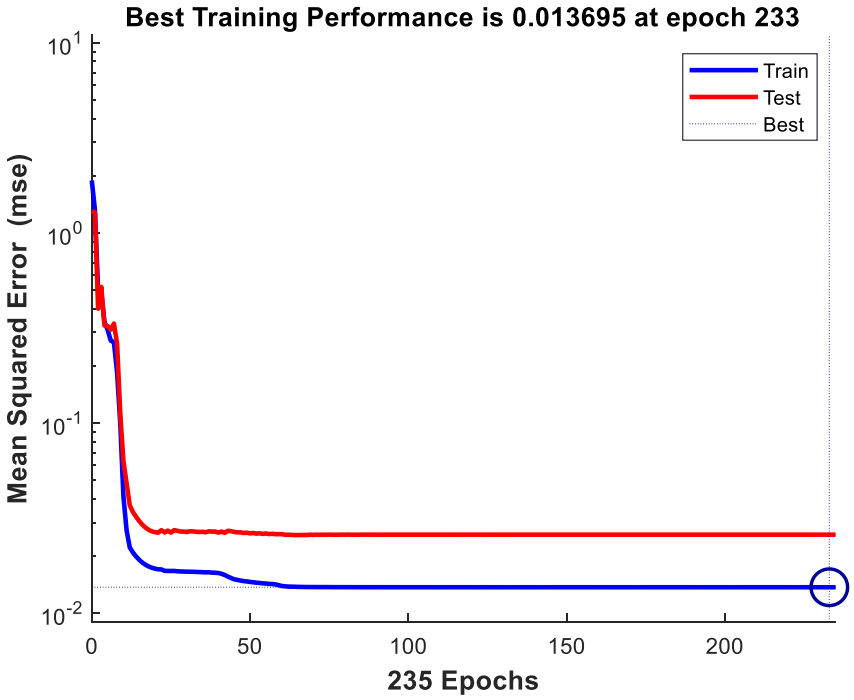

Figure 5-5. ANN training performance

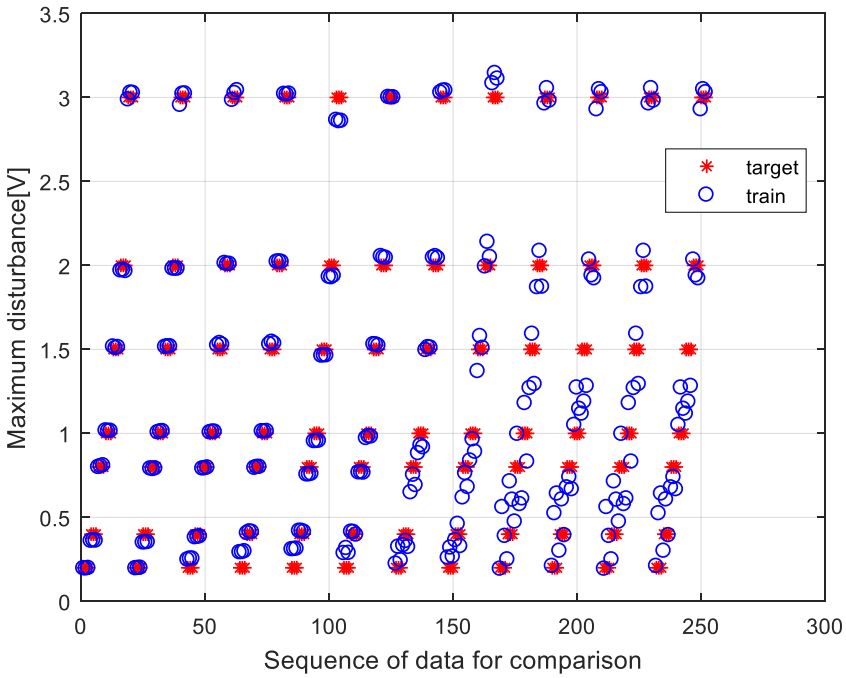

Figure 5-6. ANN training results: target (star) vs trained data (circle)

Figures 5-7 to 5-8 present the experimental results to verify the ANN method. A white gausian disturbance is intentionally injected between the control output and hardware. The standard deviation of disturbance, σ, is changed in the order of 0.5, 1, 0.333, 1, and 0.667 for every 3000 loops as shown in Fig. 4-24. The disturbance estimation results of the ANN and comparison with the PIF are presented in Figs. 5-7 and 5-8. Both the ANN and PIF methods follow original disturbance and thus, are suitable as methods for estimating the disturbance of DC motors in an NCS.

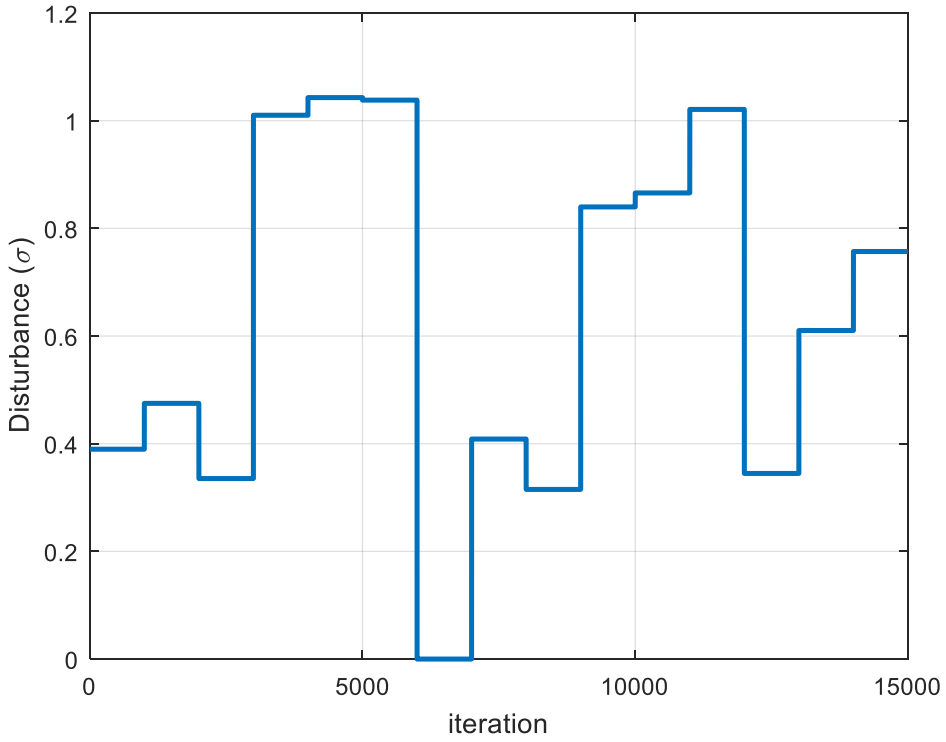

Figure 5-7. Estimated standard deviations of disturbances using the ANN method

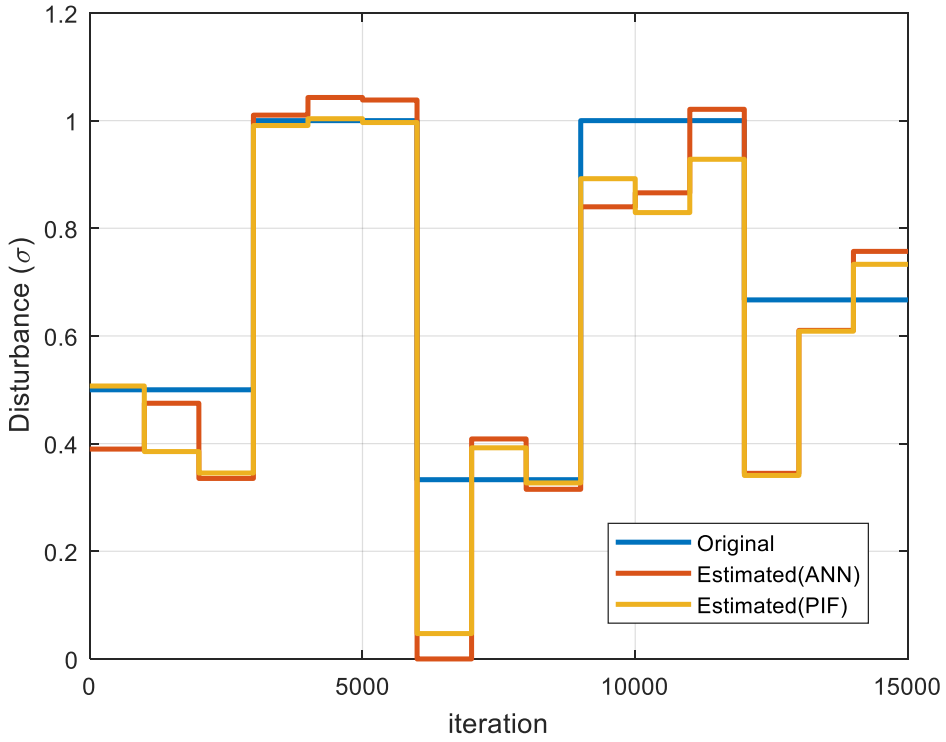

Figure 5-8. Comparison of standard deviations of disturbances between the original and the estimated using the ANN and PIF

5.2 Optimal Bandwidth Allocation for One DC Motor System Using ANN

Figures 5-9 and 5-10 present changes of sampling frequency and bandwidth utilization based on the ANN when the standard deviations of disturbances changed during 15 000 iterations as seen in Fig. 4-24. The average BU of the dynamic bandwidth allocation method using the ANN is about 13.4%.

Figure 5-11 shows the experimental results of the dynamic bandwidth allocation method using an ANN and compares the result of ANN method with those of fixed sampling periods and the PIF. Table 5.1 presents error variance and IAE comparisons between fixed and dynamic sampling periods using the ANN and PIF method. The IAE

of the ANN method is 16 773 and the average BU is 13.38%. A dynamic bandwidth allocation method such as the PIF and ANN shows best performance of 16 615 and 16 773 IAE and uses an economical BU of 13.15% and 13.38%, respectively.

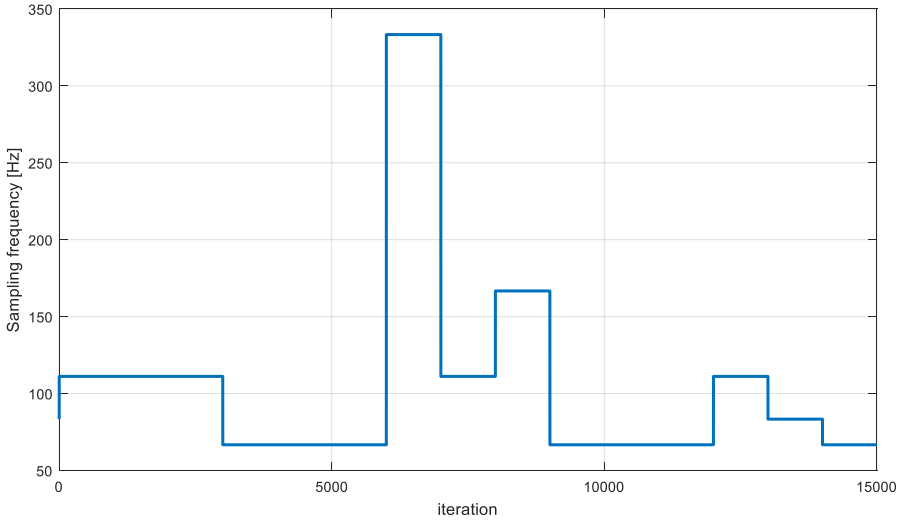

Figure 5-9. Changes of optimal sampling frequency using the ANN during 15 000 iterations when various standard deviations of disturbances were injected into the system

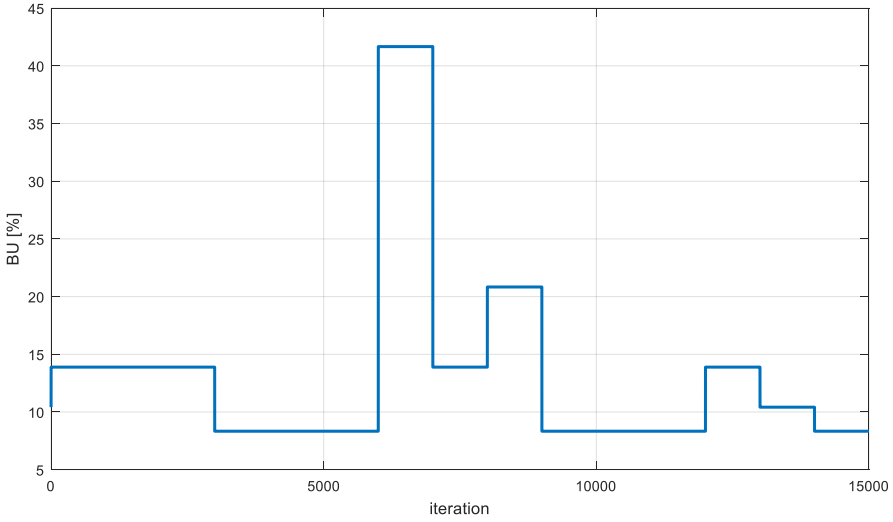

Figure 5-10. Optimal bandwidth utilization using the ANN during 15 000 iterations when various standard deviations of disturbances were injected into the system

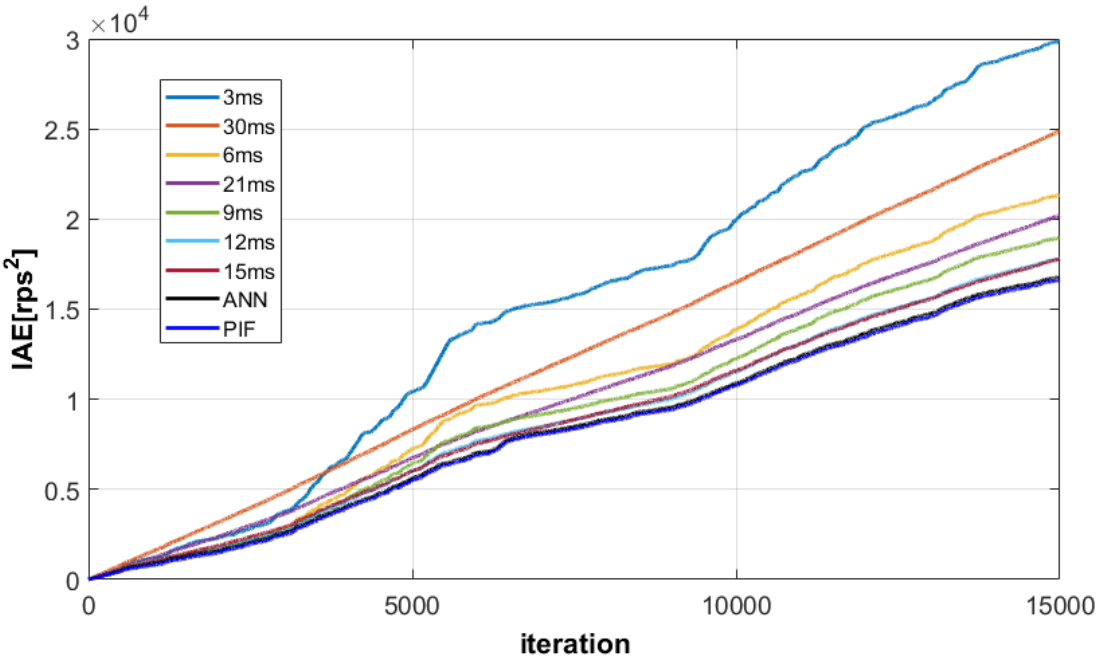

Figure 5-11. Comparison of the IAEs between fixed sampling periods (3 ms, 6 ms, 9 ms, 12 ms, 15 ms, 21 ms, and 30 ms) and the dynamic bandwidth allocation methods using the PIF and ANN, respectively

Table 5.1 Comparison of error variances and IAEs between fixed and dynamic sampling time using the PIF and ANN method, respectively

Sampling Time (ms)	3	6	9	12	15
Error variance	6.8565	3.5233	2.7608	2.3455	2.2438
IAE	29864	21326	18964	17802	17787
BU (%)	41.7	20.8	13.9	10.4	8.3
Sampling Time (ms)	21	30	PIF	ANN	
Error variance	2.6616	3.8503	2.0202	2.0530	
IAE	20195	24885	16615	16773	
BU (%)	6.0	4.2	13.15	13.38	

5.3 Optimal Bandwidth Allocation for Multi-Client System Using ANN

Unlike the single client system, the weight of each DC motor and total bandwidth utilization should be considered to find the optimal bandwidth allocation for a multi-client system. A diagram of an ANN is presented in Fig. 5-12 to determine the optimal sampling

frequency for four DC motor systems. Six inputs are the weight of each DC motor and the estimated disturbance and safety margin of BU, and the output is the optimal sampling frequencies of each DC motor. Ten neurons and two layers are used for the ANN and the Elliot-sigmoid function is chosen as an active function to take adavantage of faster calculation time. To have various data set to train and validate the ANN method, 10 752 pieces of data are obtained based using the PIF method, described in Section 4, to find the optimal sampling frequency when the standard deviation of disturbance increases from zero to 1 with a rate of increase of 0.2, the safety margin of BU increases from 0% to 60% with a rate of increase of 10%, and the weights of the DC motor systems increase from 1 to 4 with a rate of increase of 1. The bias of 1st layer are 0.384, −0.182, −2.989, 1.0318, 2.806, −60.499, 0.1580, −52.228, −41.761, and −44.019, respectively. The weights of 1st layer are 0.013, 0.002, 0.001, 0.002, 0.126, -0.591, 0.060, -0.099, 0.032, 0.032, 0.270, −0.343, 0.493, 0.225, −1.141, 0.169, −0.320, 2.298, −0.165, 0.331, −0.065, −0.068, 0.098, −0.715, −0.479, −0.216, −0.145, 1.044, 0.295, −2.171, 12.885, −9.423, −9.624, −8.953, −24.500, −20.737, −0.018, 0.018, −0.007, −0.007, −0.208, 0.291, −7.899, −8.070, −8.042, 11.053, −21.477, −17.731, −6.555, −6.249, 8.917, −6.498, −16.982, −14.274, -6.914, 9.388, −6.434, −6.770, −18.061, and −14.913, repectively. The bias of 2nd layer is −0.280, −0.567, −0.409, and −0.422, respectively. The weights of 2nd layer are 1.401, −0.023, 0.100, −0.302, −0.110, 0.636, 0.921, −0.090, −0.087, −0.099, 0.687, 0.937, 0.043, 0.587, −0.043, −0.082, 1.920, −0.082, −0.088, 0.664, 0.830, 0.028, −0.209, −0.103, −0.041, −0.076, 0.789, −0.0714, 0.667, −0.087, 0.833, 0.026, 0.042, −0.104, 0.217, −0.075, 0.795, 0.648, −0.073, and −0.082, repectively.

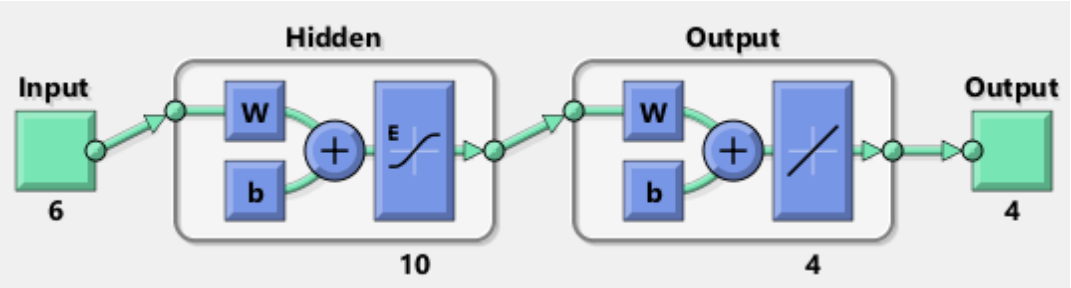

Figure 5-12. Diagram of an ANN for the optimal bandwidth allocation of four DC motor systems

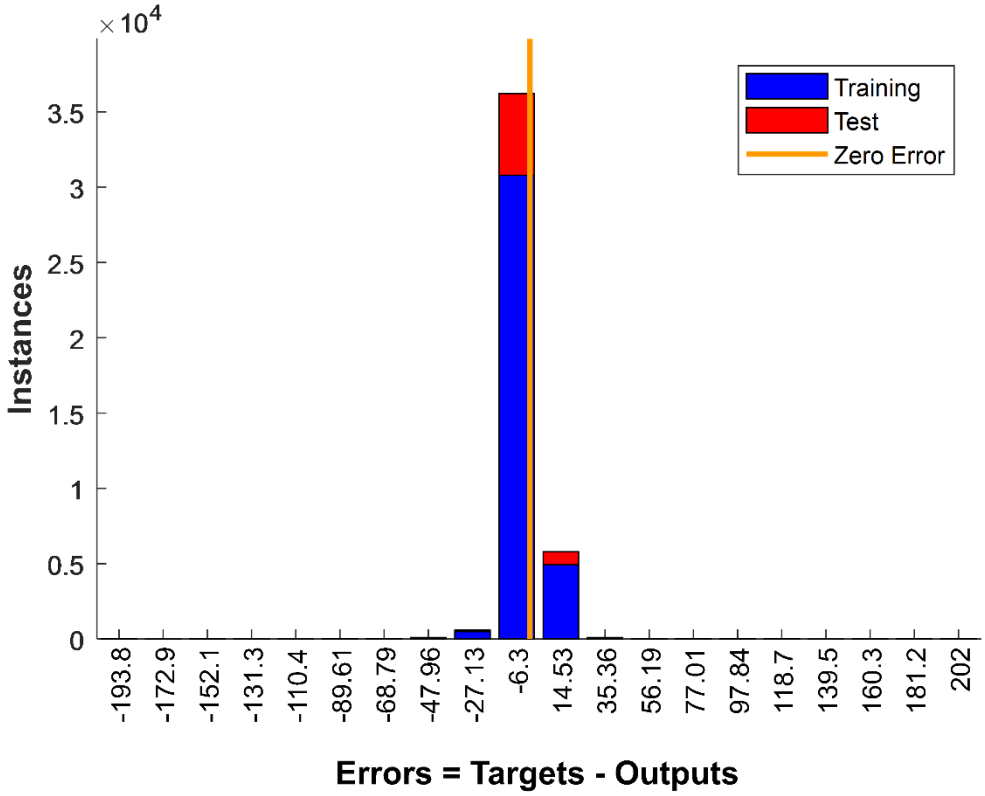

Figure 5-13. Error histogram of an ANN for the optimal bandwidth allocation of four DC motor systems

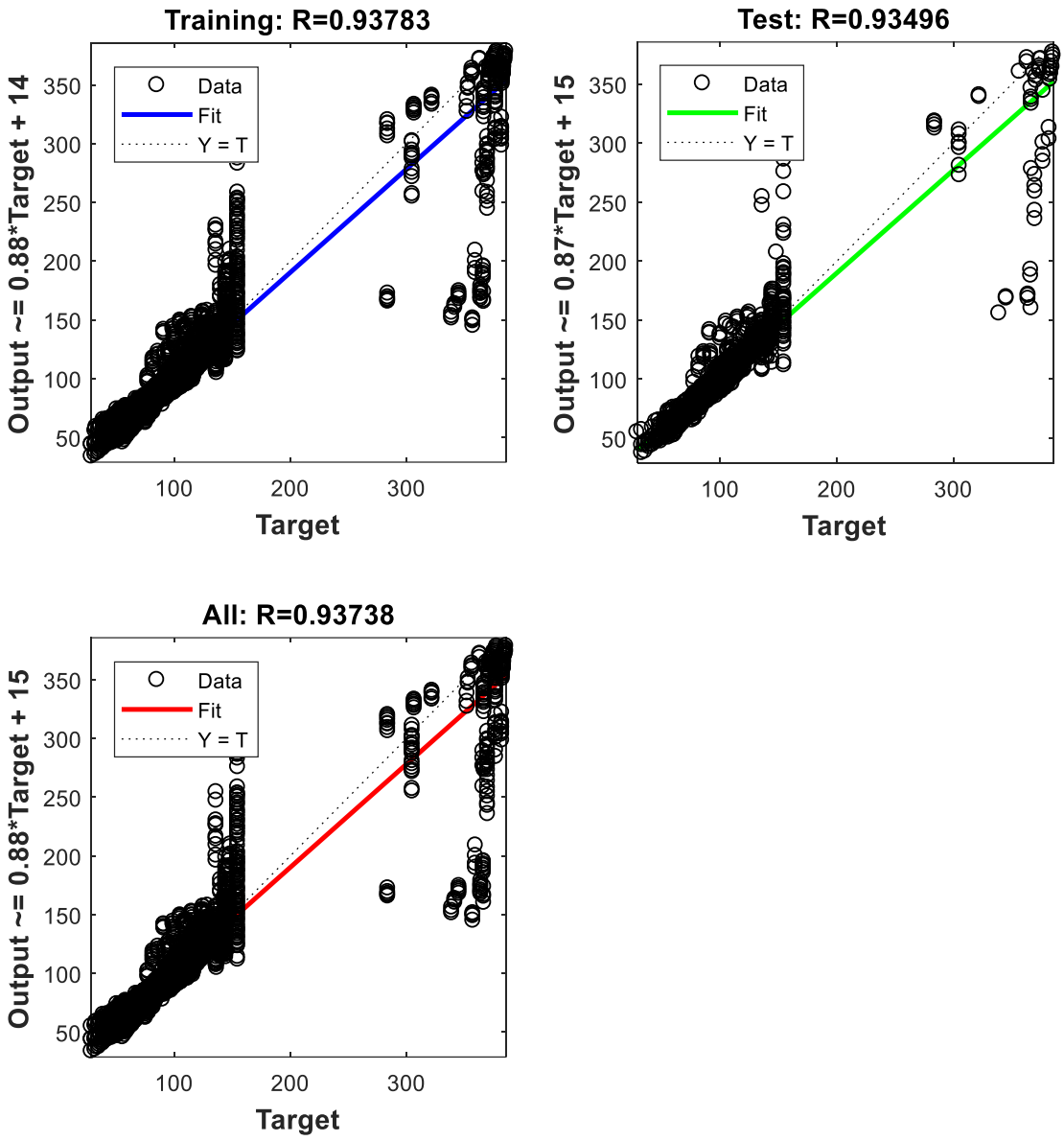

Figure 5-14. The R-squared value of the ANN to find the optimal bandwidth allocation

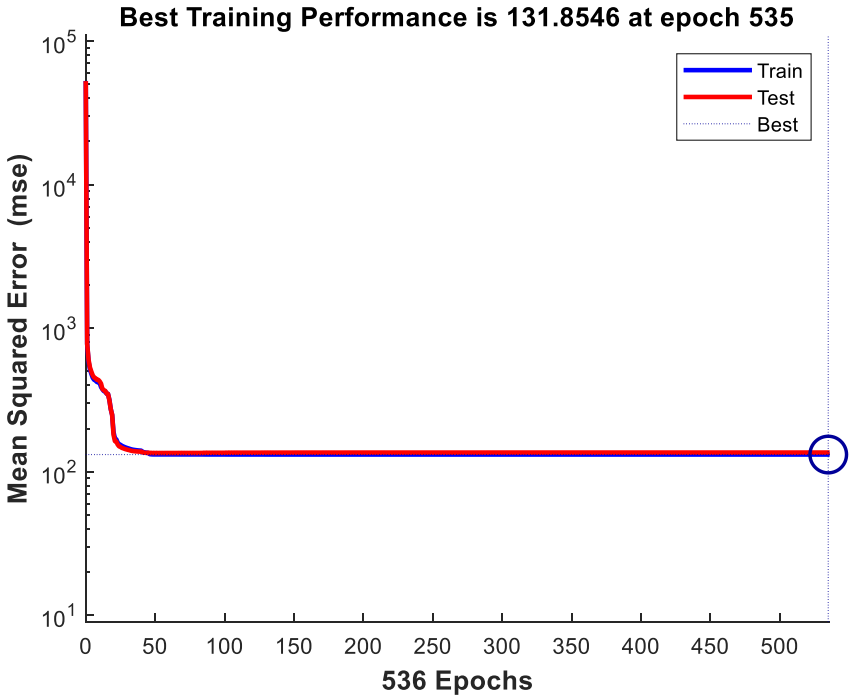

Figure 5-15. ANN training performance

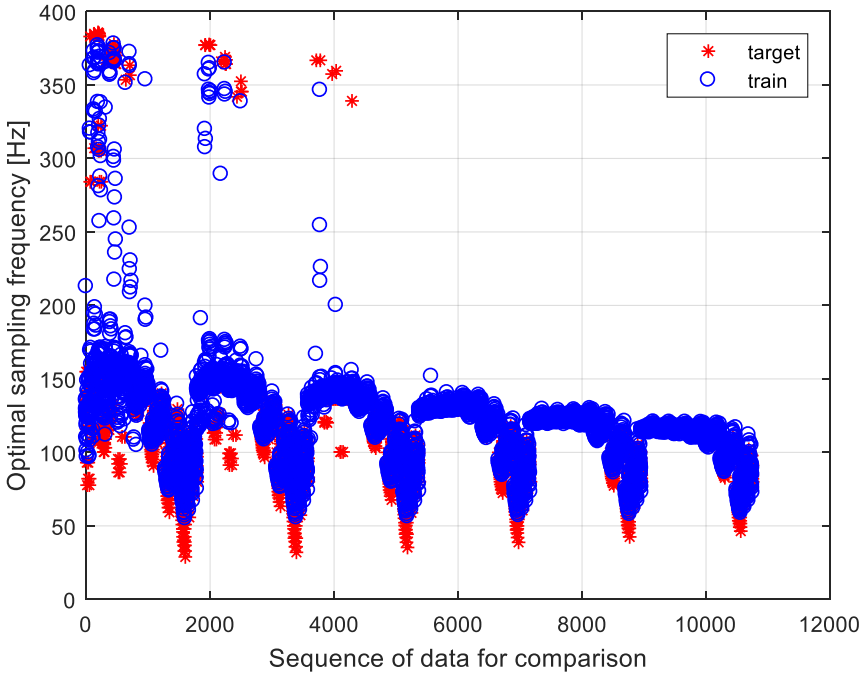

Figure 5-16. ANN training results: target (star) vs trained data (circle)

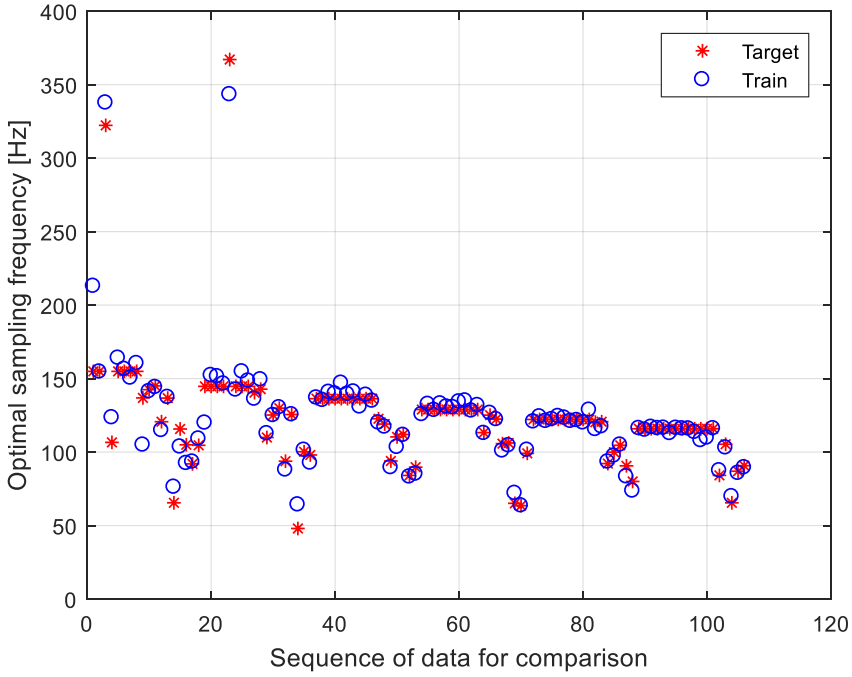

Figure 5-17. Reduced ANN training results: target (star) vs trained data (circle)

Figures 5-13 to 5-17 present the training results of an ANN to find the optimal bandwidth allocation of a four DC motor system. Most of the errors are in ±20 Hz as illustrated in Fig. 5-13, the R-squared value is 0.94 as in Fig 5-14, and the training performance of ANN is presented in Fig. 5-15. Most of the data are located lower than 170 Hz or higher than 280 Hz in Fig. 5016 because most of the optimal frequencies are located in a low frequency or a high frequency range as described in Fig. 4-2. Without disturbance the highest sampling frequencies such as 400 Hz and 333 Hz present best performance, and if the standard deviations of disturbance increase, the optimal sampling frequency drastically moves to lower frequencies such as 200 Hz, 150 Hz, 100 Hz, and 50 Hz. Since as many as 10 752 pieces of data are used, it is difficult to distinguish between

the target optimal sampling frequencies and the trained data in Fig. 5-16. Thus, a reduced version containing 106 pieces of data is presented in Fig 5-17. Trained data can follow target data with a small error. Therefore, the ANN method can be used to find optimal sampling frequency of the NCS.

5.3.1 Experimental result – Case 1*: a* ±0.8-V*disturbance,* 10% *safety margin of BU,* 1.25-ms *total time delay, weights of DC motor systems:* 4, 3, 2, *and* 1, repectively

Figure 5-18 presents the experimental results of the ANN method where a ±0.8-V disturbance is intentionally injected, the safety margin of BU is 10%, total time delay is 1.25 ms, and the weight of each DC motor is 4, 3, 2, and 1, respectively. The optimal sampling frequencies of each system are 125.3 Hz, 123.3 Hz, 121.6 Hz and 119.3 Hz and IAEs are 6844.1, 7030.2, 6975.3 and 7098.9, respectively as seen in Table 5.2. Despite a safety margin of BU of 10%, a 90% bandwidth is enough to operate in optimal sampling frequencies for four DC motor systems because four DC motor systems used a total of 62.5% BU to operate in optimal sampling frequencies. Thus, optimal sampling frequencies are almost identical in spite of the different weights of each system. Experimental results of the PIF are included in Table 5.2 as well to compare results.

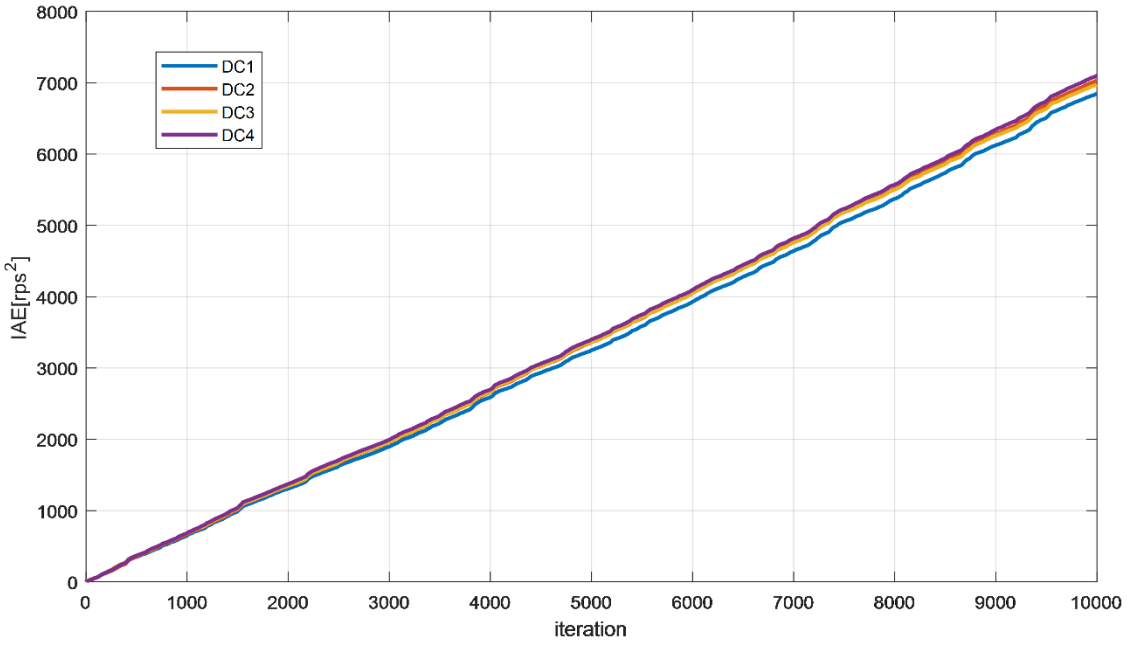

Figure 5-18. IAEs of DC motors in the case of a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, weights of DC motor systems: 4, 3, 2, and 1, repectively

Table 5.2. Comparison of optimal sampling frequencies and IAEs between the PIF and ANN method in Case 1

		DC1	DC2	DC3	DC4
Optimal sampling frequency	PIF	122.2	122.2	122.2	122.2
	ANN	125.3	123.3	121.6	119.3
IAE	PIF	6865.8	7061.6	6917.1	7108.7
	ANN	6844.1	7030.2	6975.3	7098.9

5.3.2 Experimental result – Case 2*: a* ±0.8-V *disturbance,* 10*% safety margin of BU,* 1.25-ms *total time delay, weights of DC motors:* 4*,* 3*,* 2*, and* 1*,* respectively*, with a maglev and a wheelchair robot system*

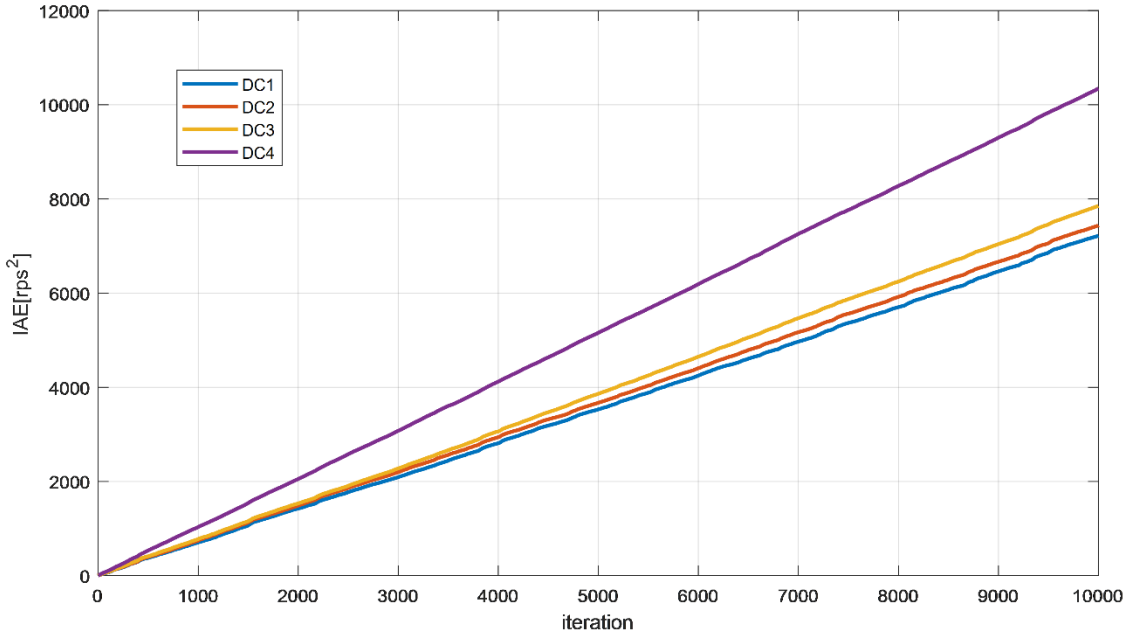

Figure 5-19. IAEs of the DC motors in the case of a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, and weights of DC motors: 4, 3, 2, and 1, respectively, with a maglev and a wheelchair robot system

Figure 5-19 presents the experimental results of an ANN method where a ±0.8-V disturbance is intentionally injected, the safety margin of BU is 10%, total time delay is 1.25 ms, and the weight of each DC motor is 4, 3, 2, 1, respectively, with a maglev and wheelchair robot systems. Since the maglev system with 3-ms sampling period requires 44% BU and the wheelchair robot uses 1% BU, only 45% of the bandwidth can be utilized. Optimal sampling frequencies of each system are 105.9 Hz, 100.2 Hz, 92.1 Hz, and 64.9 Hz and the IAEs are 7221.8, 7440.2, 7852.1 and 10 344 as shown in Table 5.3. Due to the

different weight, each system has a different optimal sampling frequency and system performance increases in the ascending order of weights. Experimental results of the PIF are included in Table 5.3 as well for comparison.

Table 5.3. Comparions of IAEs between the PIF and ANN method in Case 2

		DC1	DC2	DC3	DC4
Optimal sampling frequency (Hz)	PIF	103.7	98.7	89.7	67.9
	ANN	105.9	100.2	92.1	64.9
IAE (rps^2)	PIF	7281.7	7689.2	8002.9	9945.8
	ANN	7221.8	7440.2	7852.1	10344

5.4 Conclusion of ANN Method for NCS with Disturbance and Noise to Find Optimal Bandwidth Allocation

Unlike the nonlinear constraint optimization method using the PIF, the ANN method does not need to operate hundreds of iterations to determine optimal bandwidth allocation every time. Thus, it is more beneficial in real-time operation such as NCSs compared to the PIF method. Also, by training a well-selected data set, the estimated disturbance and optimal sampling frequency results can be close to the results of the PIF.

6. Q-LEARNING METHOD FOR THE NCS WITH DISTURBANCE AND NOISE

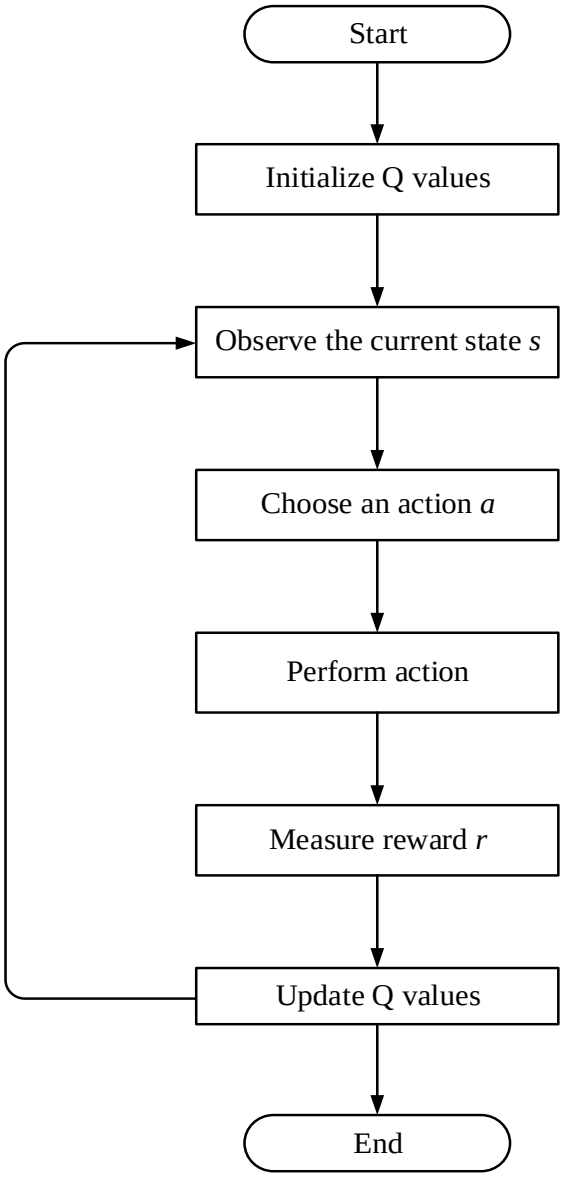

Figure 6-1. Flowchart of the Q-learning algorithm for an NCS

Figure 6-1 presents a flowchart of the Q-learning algorithm for an NCS. First, Q-values are initialized at any arbitrary values. In this dissertation, initial Q-values are set to

zero. Then, a current state s is observed. The weight of each system, safety margin of BU, and standard deviation of disturbance make up the current state. Based on the current Q-value and using an ε-greedy strategy, an action *a* (sampling frequency) in the current state *s* is selected. We first generate a random number, and if the number is greater than ε, then an "exploitation" is performed. If not, an "exploration" is done. After taking the action ***a*,** the outcome state *s´* and reward *r* is observed. Q-values are updated according to Eq. 2.4. The process is repeated until learning is stopped or a terminal state is reached.

In an NCS, if the estimated disturbance, weight of each system, and safety margin of BU are known for a current state, the optimal bandwidth can be calculated by updating Q-values and choosing an action which has a maximum Q-value. The advantage of the Q-learning algorithm is that it does not need experimental data such as an ANN or PIF to find the relationship between the amount of disturbance and the optimal sampling frequency. Without knowledge of the PIFs, the Q-learning method can find an optimal bandwidth allocation for an NCS.

6.1 Optimal Bandwidth Allocation for a DC Motor System Using Q-learning

Figure 4-24 presents how the standard deviation of disturbance, σ, was changed for the experiment. Figures 6-2 to 6-5 show how Q-values are changed for 50 000 iterations for a DC motor system with a ±1.5-V, ±3-V, ±1-V, ±2-V disturbance, respectively. Possible actions are to select a 3-ms to 30-ms sampling period with increments of 1 ms. Thus, the total number of possible actions are 28. Initial Q-values are

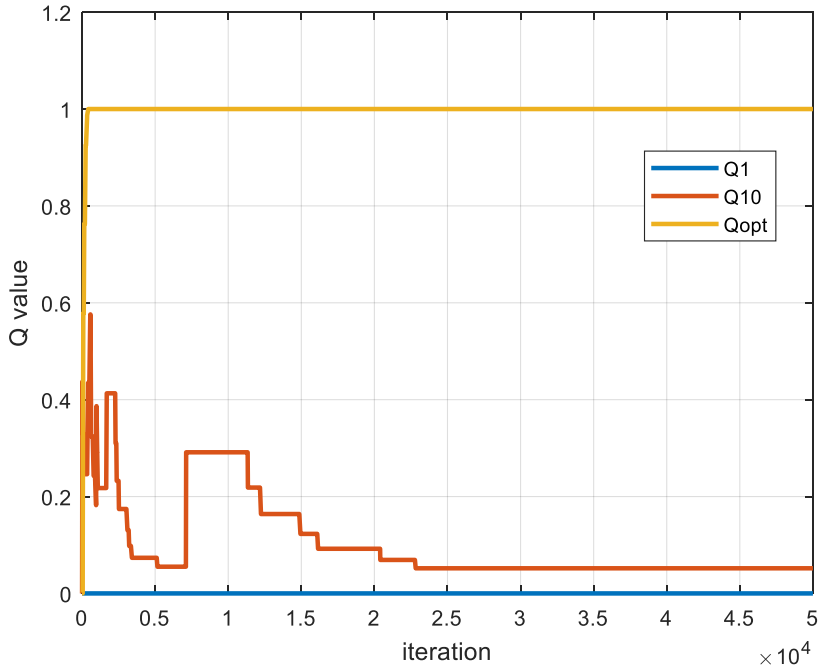

Figure 6-2. Changes in two arbitrary Q-values and an optimal Q-value for a DC motor system with a ±1.5-V disturbance

set to zero, learning rate α is 0.25, and the discount factor γ is zero because the chosen action (sampling frequency) does not change the status of the system, which consists of the weight of each DC motor system, safety margin of BU, and standard deviation of disturbance. A reward is 1 when an error variance is equal to or less than three consecutive previous results, or the reward is 0. The ε-greedy function is designated as $g_1/(g_2+k)$, where g_1 is 300, g_2 is 400, and k is the number of iterations. The ε-greedy function allows the system to explore randomly at the beginning and the ratio of exploitation increases when the number of iterations grows. The g_1 and g_2 depend on the number of actions.

Figure 6-2 presents two arbitrary Q-values and an optimal Q-value change for a DC motor system with a ±1.5-V disturbance. As iterations increased, optimal Q-values converged to 1 and others became small numbers, less than 0.1. After 50 000 iterations,

Q-values became 0, 0.0003, 0, 0.0016, 0.0042, 0.0047, 1.0000, 0.0006, 0.0044, 0.0519, 0.0020, 0.0105, 0.0006, 0.0014, 0.0001, 0.0013, 0.0010, 0.0008, 0.0658, 0, 0, 0.0020, 0.0001, 0, 0, 0.0001, 0.0003, and 0, respectively. The optimal sampling frequency was 111.1 Hz, which corresponds to the index of the highest Q-value.

Figure 6-3 presents two arbitrary Q-values and an optimal Q-value change for a DC motor system with a ±3-V disturbance. As iterations increased, optimal Q-values converged to 1 and others became small numbers, less than 0.1. After 50 000 iterations, Q-values became 0, 0, 0, 0.0001, 0.0006, 0.0001, 0.0019, 0.0033, 0.0006, 0.0375, 0.0449, 0.0009, 1.0000, 0.0011, 0.0007, 0.0014, 0.0143, 0.0002, 0.0005, 0.0008, 0.0855, 0.0008, 0.0003, 0.0001, 0, 0.0001, 0.0011, and 0.0012, respectively. The optimal sampling frequency was 66.7 Hz, which corresponds to the index of the highest Q-value.

Figure 6-4 shows two arbitrary Q-values and an optimal Q-value change for a DC motor system with a ±1-V disturbance. As iterations increased, optimal Q-values converged to 1 and others became small numbers less than 0.1. After 50 000 iterations, Q-values became 0.0001, 0.0049, 0.0039, 0.0012, 0.0067, 1.0000, 0.0027, 0.0057, 0.0117, 0.0109, 0.0001, 0.0288, 0.0001, 0.0015, 0.0106, 0.0041, 0.0392, 0, 0, 0.0633, 0.0781, 0, 0.0004, 0, 0, 0, 0, and 0, respectively. The optimal sampling frequency is 125 Hz, which corresponds to the index of the highest Q-value.

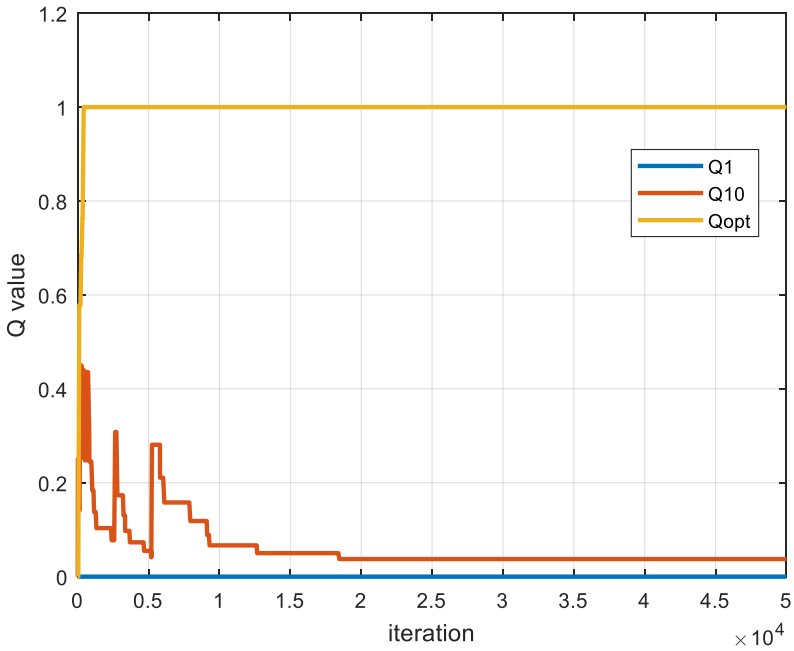

Figure 6-3. Changes in two arbitrary Q-values and an optimal Q-value for a DC motor system with a ±3-V disturbance

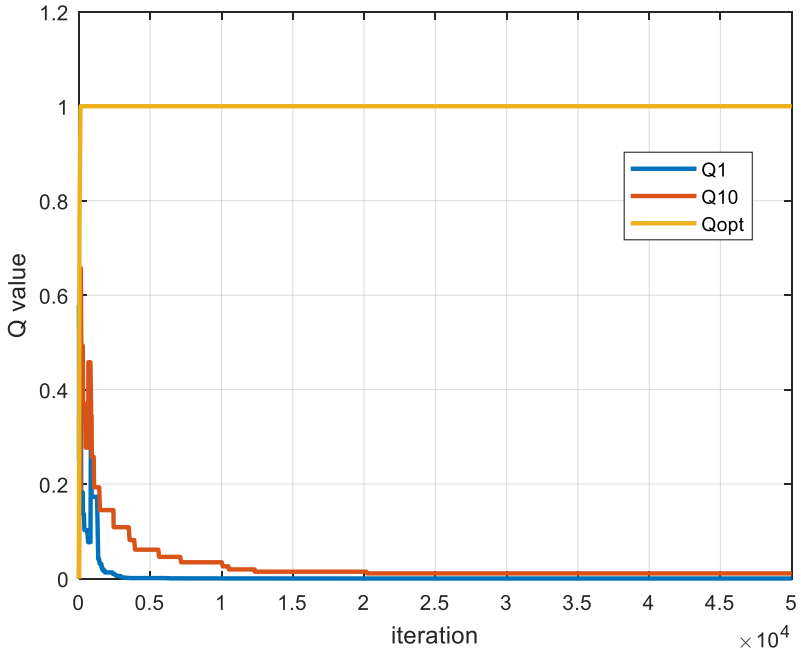

Figure 6-4. Changes in two arbitrary Q-values and an optimal Q-value for a DC motor system with a ±1-V disturbance

Figure 6-5 shows two arbitrary Q-values and an optimal Q-value change for a DC motor system with a ±2-V disturbance. As the iterations increased, the optimal Q-value converged to 1 and others became small numbers, less than 0.1. After 50 000 iterations, the Q-values became 0.0001, 0.0001, 0, 0.0011, 0.0020, 0.0003, 0.0032, 1.0000, 0.0007, 0.0157, 0.0001, 0.0196, 0.0004, 0.0020, 0.0005, 0.0003, 0.0006, 0.0003, 0.0139, 0.0014, 0.0001, 0.0410, 0, 0.0025, 0.0005, 0.0001, 0.0002, and 0.0003, respectively. The optimal sampling frequency was 100 Hz, which corresponds to the index of the highest Q-value.

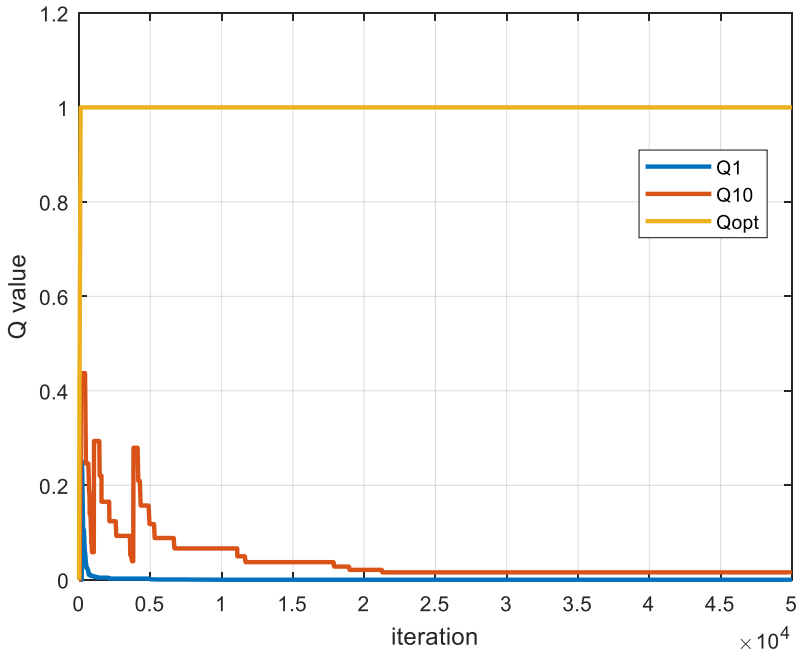

Figure 6-5. Changes in two arbitrary Q-values and an optimal Q-value for a DC motor system with a ±2-V disturbance

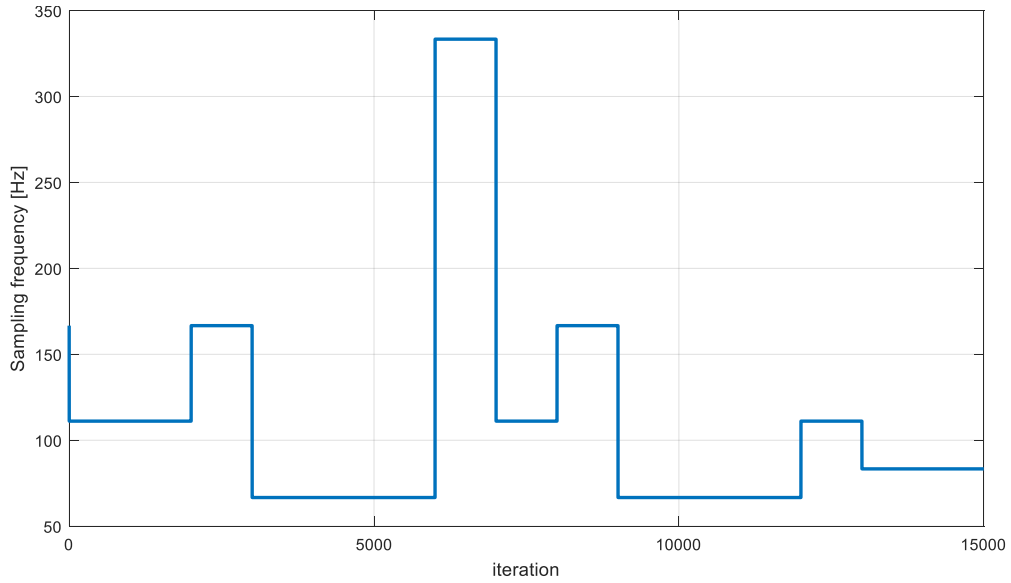

Figure 6-6. Changes of optimal sampling frequencies using the Q-learning during 15 000 iterations when various standard deviations of disturbances were injected into the system

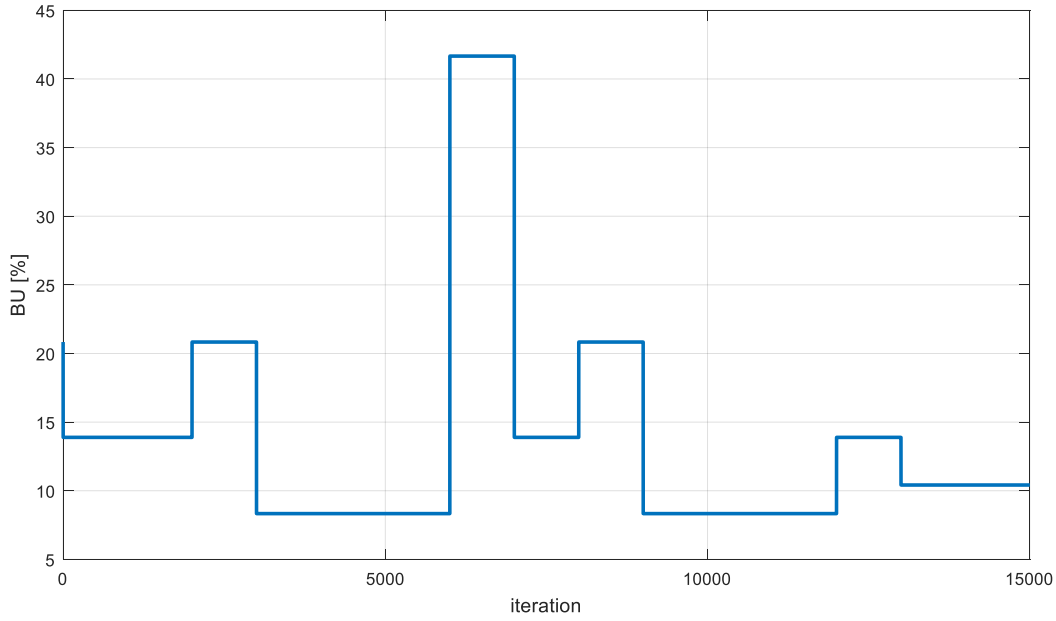

Figure 6-7. Optimal bandwidth utilization using the Q-learning during 15 000 iterations when various standard deviations of disturbances were injected into the system

Figures 6-6 and 6-7 present changes of sampling frequencies and BU using Q-learning when the standard deviations of disturbances changed during 15 000 iterations as shown in Fig. 4-24. In this experiment, we limited possible sampling frequencies to 333 Hz, 167 Hz, 111 Hz, 83 Hz, 67 Hz, 56 Hz, 48 Hz, and 33 Hz to compare the results with previous results from the PIF and ANN. After, the standard deviations of disturbances was estimated using either the PIF or ANN method, the Q-learning algorithm was used to find the optimal sampling frequency. First, estimated disturbance is sent to the server. The server identifies the current status and chooses the optimal action which has the highest Q-value. The average BU of the dynamic bandwidth allocation method using the Q-learning method is about 13.98%.

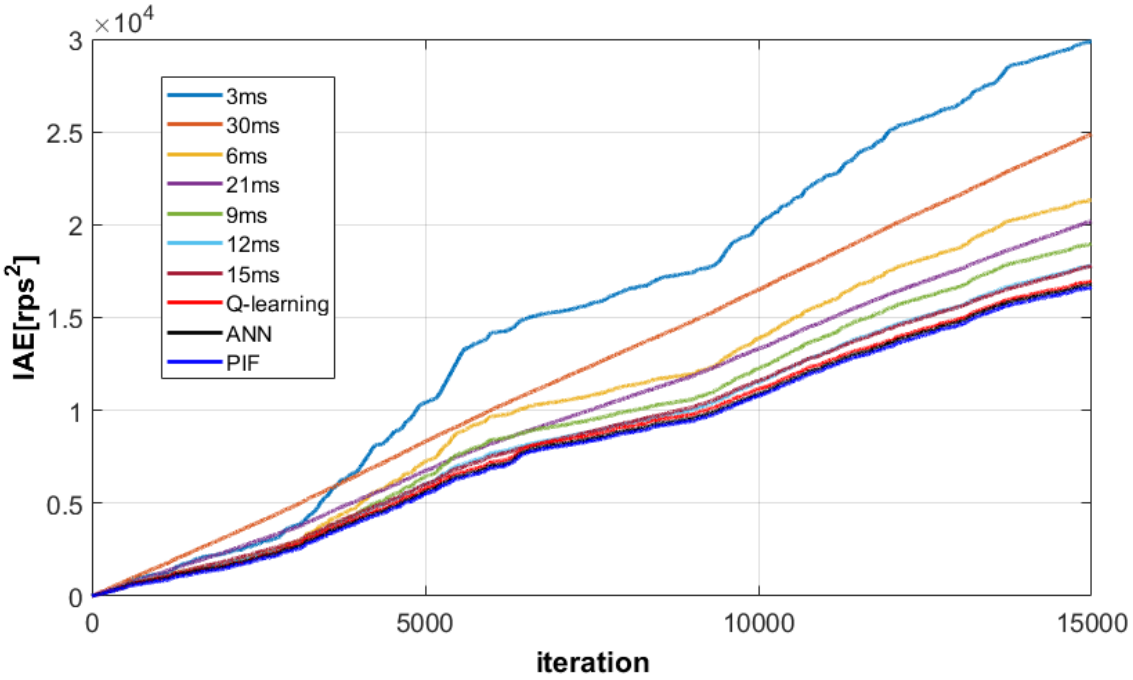

Figure 6-8. Comparison of the IAEs between fixed sampling periods (3 ms, 6 ms, 9 ms, 12 ms, 15 ms, 21 ms, and 30 ms) and the dynamic bandwidth allocation method susing the PIF, ANN, and Q-learning

Table 6.1. Comparison of error variances and IAEs between fixed and dynamic sampling periods using Q-learning, PIF and ANN method, repectively

Sampling Time (ms)	3	6	9	12	15
Error variance	6.8565	3.5233	2.7608	2.3455	2.2438
IAE	29864	21326	18964	17802	17787
BU (%)	41.7	20.8	13.9	10.4	8.3
Sampling Time (ms)	21	30	PIF	ANN	Q-learning
Error variance	2.6616	3.8503	2.0202	2.0530	2.0916
IAE	20195	24885	16615	16773	16945
BU (%)	6.0	4.2	13.15	13.38	13.98

Figure 6-8 presents the experimental results of the dynamic bandwidth allocation method using Q-learning to compare it with fixed sampling periods, the PIF and the ANN methods. Table 6.1 shows the exact error variances, IAEs, and BUs to allow comparisons between the fixed and dynamic sampling periods using the Q-learning, ANN, and PIF methods. The IAE of Q-learning is 16 945 and the BU is 13.98%. Compared to fixed sampling periods, a dynamic bandwidth allocation method such as Q-learning, PIF, or ANN shows best performance of 16 945, 16 615, and 16 773 IAEs and uses economical BUs of 13.98%, 13.15%, and 13.38%, respectively.

6.2 Optimal Bandwidth Allocation for a Multi-Client System Using Q-learning

6.2.1 Experimental result – Case 1*: a* ±0.8-V *disturbance,* 10% *safety margin of BU,* 1.25-ms *total time delay, weights of DC motor systems:* 4, 3, 2, *and* 1, repectively

Unlike the single-client system, the weight of each DC motor and total bandwidth utilization should be considered to find the optimal bandwidth allocation for a multi-client system. Possible actions for each DC motor system were set to select a sampling frequency

among 333 Hz, 250 Hz, 200 Hz, 167 Hz, 143 Hz, 125 Hz, 111 Hz, 100 Hz, 91 Hz, 83 Hz, 77 Hz, 71 Hz, 67 Hz, 56 Hz, 48 Hz, 42 Hz, 37 Hz, and 33 Hz. Thus, a total of 104 976 (=18×18×18×18) actions were available. However, available BU was considered at the beginning and possible actions were reduced to 98 561 after the actions exceeding the limitation of BU were eliminated. This process removes unnecessary calculations during Q-learning based on each scenario. Initial Q-values were set to zero, the learning rate α was 0.25, and the discount factor γ was zero because the chosen action does not change the status of the system such as the weight of each system, safety margin of BU, or standard deviation of disturbance. Too low a learning rate does not progresses and too high a learning rate causes instability and does not converges. Thus, to select a proper learning rate is important. In this dissertation, the learning rate, 0.25, was selected after dozens of simulation. The reward is 1 when the error variance is equal to or less than twenty consecutive previous results or the reward is 0. The ε-greedy function was designated as $g_1/(g_2+k)$, where g_1 is 1 500 000, g_2 is 2 000 000, and k is the number of iterations. Since more than 50 times actions are available compared to cases in section 6.1, much larger number of g_1 and g_2 are selected. Using the ε-greedy function, the system explores randomly at the beginning and increases the ratio of exploitation when the number of iterations grows.

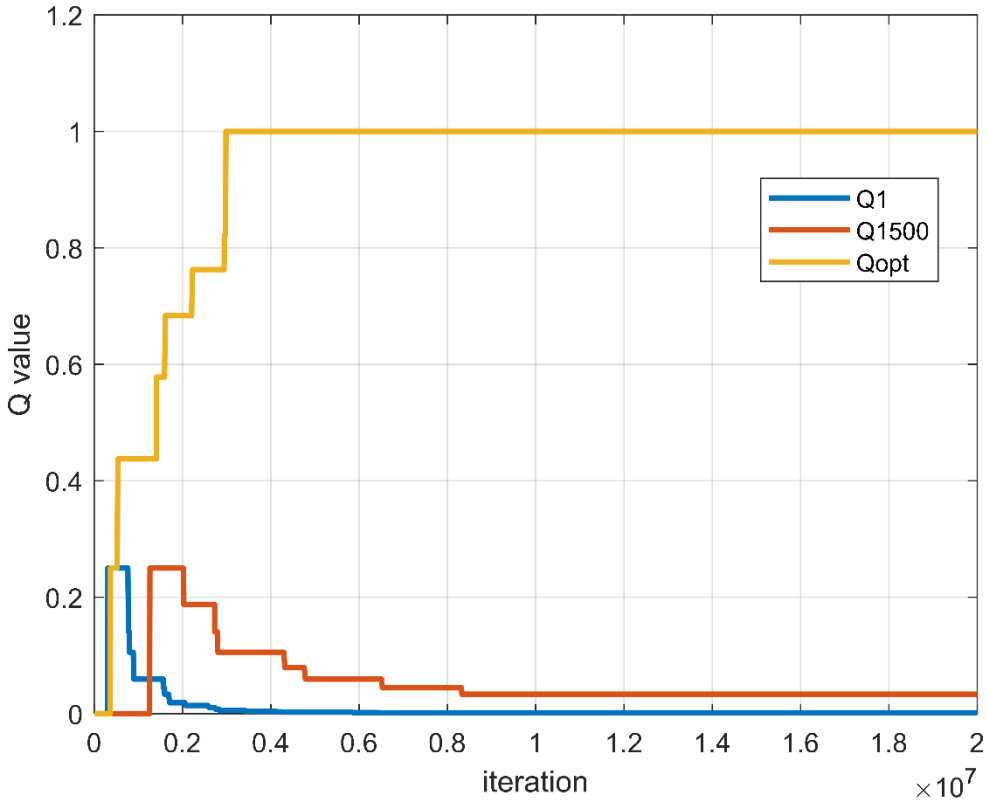

Figure 6-9. Changes in two arbitrary Q-values and an optimal Q-value for four DC motor systems with a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, weights of DC motor systems: 4, 3, 2, and 1, repectively

Figure 6-9 shows two arbitrary Q-values and an optimal Q-value change for a four DC motor system with a ±0.8-V disturbance, a 10% safety margin of BU, and a 4, 3, 2, 1 weight for each motor, respectively. As iterations increased, the optimal Q-value converged to 1 and others became small numbers, less than 0.1. The optimal sampling frequencies for each DC motor were 125 Hz, 125 Hz, 125 Hz, and 125 Hz, respectively, which corresponds to the index of the highest Q-value.

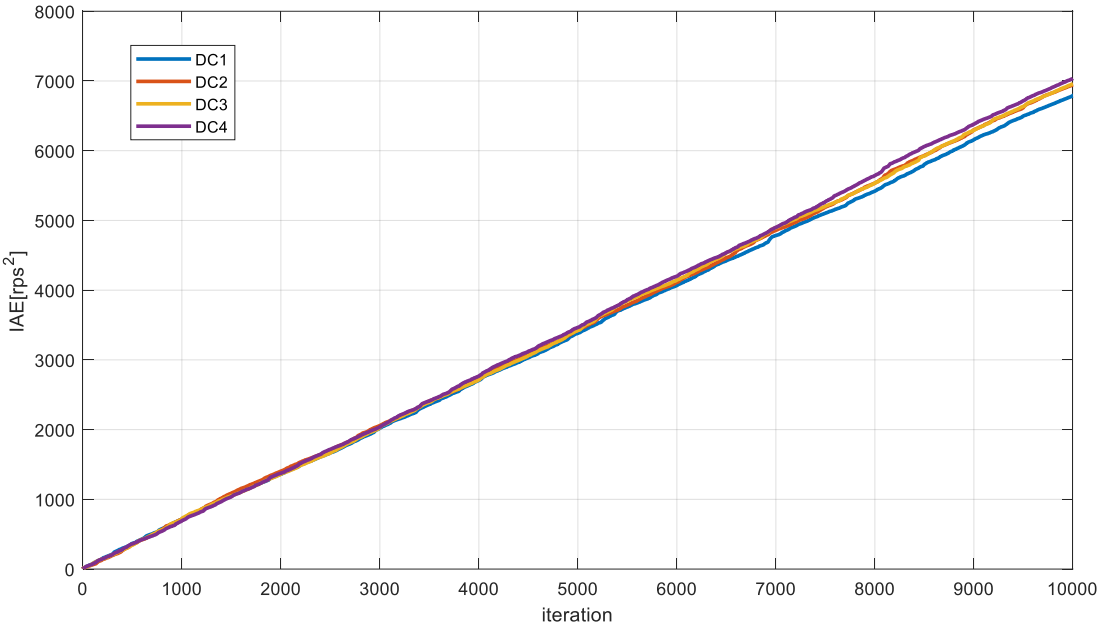

Figure 6-10. IAEs of four DC motor systems with a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, weights of DC motor systems: 4, 3, 2, and 1, repectively

Figure 6-10 presents the experimental results of the Q-learning algorithm where a ±0.8-V disturbance was intentionally injected, the safety margin of BU was 10%, the total time delay of each DC motor system was 1.25 ms, and the weight of each DC motor was 4, 3, 2, and 1, respectively. The optimal sampling frequencies of each system are 125 Hz, 125 Hz, 125 Hz, and 125 Hz and the IAEs are 6793.0, 6948.3, 6961.6, and 7035.6 as seen in Table 6.2. Despite a safety margin of 10%, an available 90% bandwidth is enough to operate the system at an optimal sampling frequency for the four DC motors since the total BU is 62.4%. Thus, the optimal sampling frequencies of the all DC motors are identical despite the different weights of each motor. Experimental results of the ANN and PIF are also included in Table 6.2 to compare them with those of the Q-learning.

Table 6.2. Comparison of IAEs and BUs between the PIF, ANN, and Q-learning method in Case 1

		DC1	DC2	DC3	DC4
Optimal sampling frequency (Hz)	PIF	122.2	122.2	122.2	122.2
	ANN	125.3	123.3	121.6	119.3
	Q-learning	125	125	125	125
IAE	PIF	6865.8	7061.6	6917.1	7108.7
	ANN	6844.1	7030.2	6975.3	7098.9
	Q-learning	6793.0	6948.3	6961.6	7035.6
BU (%)	PIF	15.3	15.3	15.3	15.3
	ANN	15.7	15.4	15.2	14.9
	Q-learning	15.6	15.6	15.6	15.6
Total BU (%)	PIF	61.2			
	ANN	61.2			
	Q-learning	62.4			

6.2.2 Experimental result – Case 2*: a* ±0.8-V *disturbance,* 10*% safety margin of BU,* 1.25-ms *total time delay, weights of DC motors:* 4*,* 3*,* 2*, and* 1*,* respectively*, with a maglev and a wheelchair robot system*

Possible actions for each DC motor system were set to select a sampling frequency among 333 Hz, 250 Hz, 200 Hz, 167 Hz, 143 Hz, 125 Hz, 111 Hz, 100 Hz, 91 Hz, 83 Hz, 77 Hz, 71 Hz, 67 Hz, 56 Hz, 48 Hz, 42 Hz, 37 Hz, and 33 Hz as in Case 1. Thus, a total of 104 976 actions were available. However, the available BU was considered at the beginning and the possible actions were reduced to 33 833 after the actions exceeding the limitation of BU were eliminated. Initial Q-values were set to zero, the learning rate α was 0.25, and the discount factor γ was zero because the chosen action does not change the

weight of the system, safety margin of BU, or standard deviation of disturbance. The reward is 1 when the MSE is equal to or less than twenty consecutive previous results or the reward is 0. The ε-greedy function was designated as $g_1/(g_2+k)$, where g_1 is 1 500 000, g_2 is 2 000 000, and k is the number of iterations. The ε-greedy function allows the system to explore randomly at the beginning and the ratio of exploitation increases when the number of iterations grows.

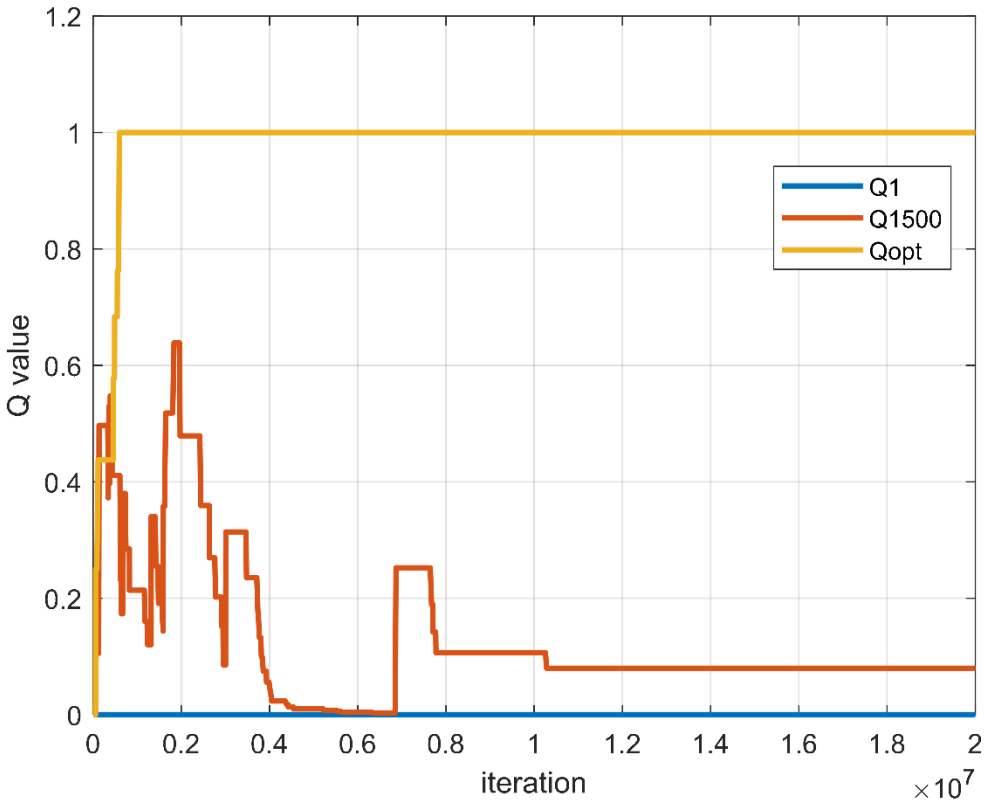

Figure 6-11. Changes in two arbitrary Q-values and an optimal Q-value for four DC motor systems with a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, weights of DC motor systems: 4, 3, 2, and 1, repectively, with a maglev and a wheelchair robot system

Figure 6-11 shows two arbitrary Q-values and an optimal Q-value change for DC motor systems with a ±0.8-V disturbance, a 10% safety margin of BU, and a 4, 3, 2, 1 weight for each DC motor system, respectively, with a maglev and a wheelchair robot system. As iterations increased, the optimal Q-value converged to 1 and others became small numbers, less than 0.1. The optimal sampling frequency for each DC motor system was 100 Hz, 100 Hz, 91 Hz, and 67 Hz, respectively which corresponds to the index of the highest Q-value.

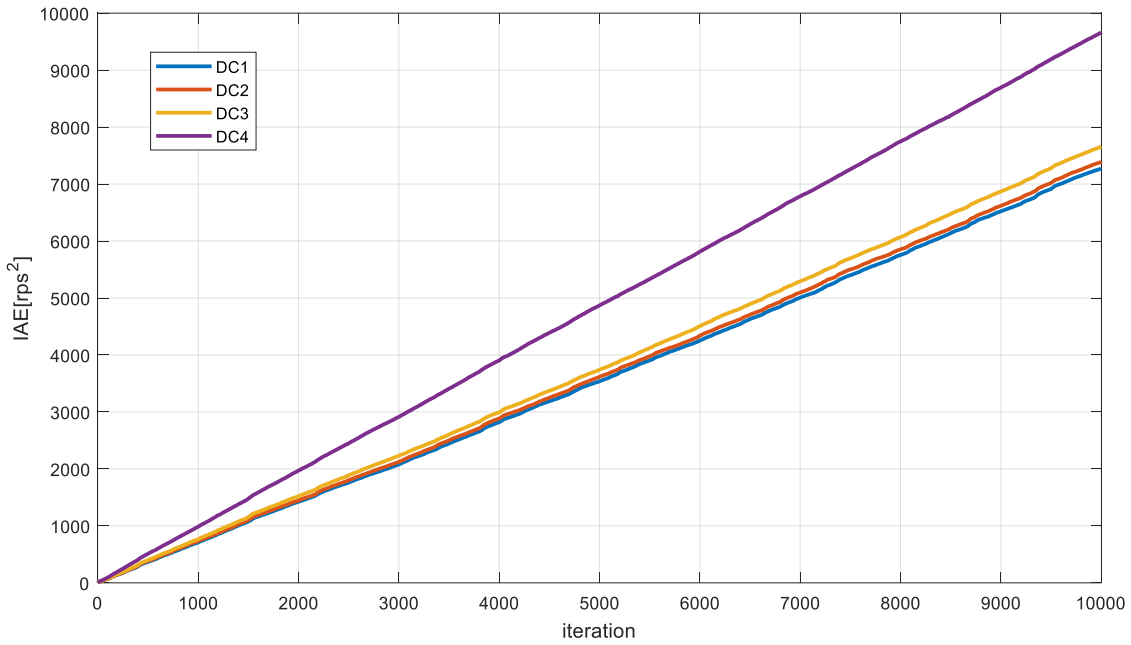

Figure 6-12. IAEs of four DC motor systems with a ±0.8-V disturbance, 10% safety margin of BU, 1.25-ms total time delay, weights of DC motor systems: 4, 3, 2, and 1, repectively, with a maglev and a wheelchair robot system

Table 6.3. Comparison of IAEs and BUs between the PIF, ANN, and Q-learning method in Case 2

		DC1	DC2	DC3	DC4
Optimal sampling frequency (Hz)	PIF	103.7	98.7	89.7	67.9
	ANN	105.9	100.2	92.1	64.9
	Q-learning	100	100	91	67
IAE	PIF	7281.7	7689.2	8002.9	9945.8
	ANN	7221.8	7440.2	7852.1	10344
	Q-learning	7275.5	7394.6	7660.5	9666.3
BU (%)	PIF	13.0	12.3	11.2	8.5
	ANN	13.2	12.5	11.5	8.1
	Q-learning	12.5	12.5	11.4	8.4
Total BU (%)	PIF	90			
	ANN	90.3			
	Q-learning	89.8			

Figure 6-12 presents the experimental results of a Q-learning algorithm where a ±0.8-V disturbance was intentionally injected, the safety margin of BU was 10%, total time delay is 1.25 ms, and the weight of each DC motor was 4, 3, 2, and 1, respectively, with a maglev and a wheelchair robot system. Since the maglev system with a 3-ms sampling period requires 44% BU and the wheelchair robot with 300-ms sampling period uses 1% BU, only 45% of the bandwidth can be utilized. The optimal sampling frequencies for each system were 100 Hz, 100 Hz, 91 Hz, and 67 Hz and the IAEs were 7275.5, 7394.6, 7660.5, and 9666.3 as shown in Table 6.3. Due to the weight, each system had a different optimal sampling frequencies and system performance increased in the order of the higher weight number. The total BU using the Q-learning algorithm was 89.8% including four

DC motor systems, a maglev, and a wheelchair robot system. Experimental results for the PIF and ANN are also included in Table 6.3 for comparison.

6.3 Conclusion: Q-learning Method for NCS with Disturbance and Noise to Find the Optimal Bandwidth Allocation

Unlike the PIF and ANN methods, the Q-learning algorithm does not need experimental data to find the optimal sampling frequency and does not have to model the PIFs. The Q-learning method can guide an optimal bandwidth allocation for an NCS without knowledge of PIFs after the standard deviations of disturbances are estimated using the PIF or ANN by updating the Q-values. Optimal sampling frequencies based on the Q-learning algorithm are close to those obtained by using the PIF and ANN. Thus, the Q-learning algorithm is another good solution for finding the optimal bandwidth.

7. CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

7.1 Conclusions

Although NCSs have many benefits such as tele-operation, reduced weight, and easy maintenance, NCSs also have some challenges such as disturbance, noise, bandwidth limitations, time delays, and package dropout. A higher sampling frequency does not guarantee better performance in NCSs, and each system has different specifications and requirements to control.

This dissertation presented three possible solutions to these problems. Exponential and 4^{th}-order polynomial functions were formulated to describe the system performance versus the sampling frequency with fixed disturbances and noises. However, the exponential approximations require more calculation time, and the 2^{nd}-order polynomials have more deviations from experimental data than the 4^{th}-order polynomials. Thus, a 4^{th}-order polynomial approximation was chosen in this research to find the optimal sampling frequency. The relationship between the standard deviations of disturbances and the performance of a system with a fixed sampling frequency could be easily approximated by using a 2^{nd}-order polynomial. Thus, the relationship between the system performance and sampling frequency and the standard deviations of disturbances could be approximated using a 6^{th}-degree polynomial with two different variables such as the sampling frequency and the disturbance. When the error variance was calculated and the sampling frequency for the experiment was identified, the standard deviations of disturbances were estimated using the PIF, which presented the relationship between disturbance and system performance with a fixed sampling frequency. After the standard

deviations of estimated disturbances are obtained, the optimal sampling frequency for a DC motor system could be determined using the PIF. A dynamic bandwidth-allocation method was proposed and experimental results were provided for the single-server and single-client DC motor system. When the standard deviations of estimated disturbancse, the safety margin of BU, the weight of each DC motor system, and the total time delay of each system were identified, the optimal sampling frequency could be determined by solving the nonlinear constraint optimization using the SQP methods. Experimental results proved the validity of this method.

With experimental data, an ANN could infer the standard devation of disturbance and noise according to the sampling frequency and error variance as inputs. To validate the proposed method, the actual injected disturbance and estimated disturbance using the ANN were compared. The dynamic bandwidth allocation for a single-server and single-client system was suggested. It was proved by experimental results, and the results were compared with the fixed sampling periods and the PIF method. Optimal bandwidth allocation using the ANN for a multi-server and multi-client system with disturbance, noise, safety margin of BU, and weight of each system was proposed. Unlike the nonlinear constraint optimization method using the PIF, the ANN method did not need to perform hundreds of iterations to solve the nonlinear constraint optimization problem to find the optimal bandwidth allocation every time for a multi-server, multi-client system. Thus, the ANN is more appropriate in real time operations such as NCSs compared to the PIF method. Also, by training a well-selected data set, the estimated disturbance and optimal

sampling frequency showed comparable results as the nonlinear constraint optimization method using the PIF.

A Q-learning algorithm does not need experimental data to obtain optimal sampling frequency and does not have to model the PIF in contrast with the PIF and ANN methods. Q-learning can be a guide to an optimal bandwidth allocation method for an NCS without knowledge of the PIFs after the standard deviations of disturbances are estimated using the PIF or ANN by updating Q-values using the Bellman equation as shown in Eq. 2.4. To validate a dynamic bandwidth allocation using the Q-learning algorithm for a single-server, single-client system, experimental data were compared with fixed sampling frequencies and the PIF and ANN. Experimental results using the Q-learning algorithm presented better performance than that with fixed sampling frequencies, which is close to the results of the PIF and ANN. Optimal bandwidth allocation using Q-learning for a multi-server, multi-client system with disturbance and noise, where the safety margin of BU and the weight of each system were identified, was also suggested and results were compared with the PIF and ANN. The experimental results verified that Q-learning algorithm was another good solution to find the optimal bandwidth allocation.

7.2 Suggestions for Future Work

In this dissertation, three solutions were proposed to find the optimal bandwidth allocation for NCSs with disturbances and noises. Several possible further research directions are explored in this section.

The plant input disturbance and the sensor output noise are assumed to be zero-mean WGN. The standard deviation of sensor output noise was assumed to be small and fixed in this dissertation. Thus, the sensor noise effect was included in the error variance of disturbance when the injected input disturbance was zero. NCSs with other types and amounts of disturbance or noise such as a constant value, pink noise, grey noise, and a mixture of them should be studied. In this dissertation, the optimal bandwidth allocation for four identical DC motor systems was studied. However, an NCS with different systems having different characteristics should also be studied. Other regression algorithms and reinforcement learnings can be considered to find the optimal bandwidth allocations, and the results should be compared with those presented in this dissertation.

REFERENCES

[1] D. Evans, “The Internet of things - How the next evolution of the Internet is changing everything,” Cisco Internet Business Solutions Group (IBSG), white paper, pp. 1-11, Apr. 2011.

[2] S. Babar, P. Mahalle, A. Stango, N. Prasad, and R. Prasad, “Proposed security model and threat taxonomy for the Internet of Things (IoT),” in *International Conference on Network Security and Applications*, Springer, Berlin, Heidelberg, July 2010, pp. 420–429.

[3] C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, and S. Sarma, “The Internet of Things: First International Conference,” in *Proceedings of IoT 2008*, vol. 4952. Springer, Zurich, Switzerland, Mar. 2008

[4] I. Gronbaek, “Architecture for the Internet of Things (IoT): API and interconnect,” in *Proceedings of 2008 Second International Conference on Sensor Technologies and Applications*, Aug. 2008, pp. 802–807.

[5] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[6] M. Yun and B. Yuxin, “Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid,” in *Proceedings of 2010 International Conference on Advances in Energy Engineering*, June 2010, pp. 69–72.

[7] K.-E. Arzen, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *Proceedings of the 39th IEEE Conference on Decision and Control,* Sydney, Australia, vol. 5, pp. 4865–4870, Dec. 2000.

[8] R. A. Gupta and M.-Y. Chow, "Networked control system: overview and research trends," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 7, pp. 2527–2535, Jul. 2010.

[9] Y. Halevi and A. Ray, "Integrated communication and control systems: Part I—Analysis," *Journal of Dynamic Systems, Measurement, and Control,* vol. 110, no. 4, pp. 367–373, Dec. 1988.

[10] A. Ray and Y. Halevi, "Integrated communication and control systems: Part II—Design considerations," *Journal of Dynamic Systems, Measurement, and Control,* vol. 110, no. 4, pp. 374–381, Dec. 1988.

[11] L. Zhang, H. Gao, and O. Kaynak, "Network-induced constraints in networked control systems—A survey," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 403–416, Feb. 2013.

[12] W. S. Wong and R. W. Brockett, "Systems with finite communication bandwidth constraints. II. Stabilization with limited information feedback," *IEEE Transactions on Automatic Control,* vol. 44, no. 5, pp. 1049–1053, May 1999.

[13] W. S. Wong and R. W. Brockett, "Systems with finite communication bandwidth constraints. I. State estimation problems," *IEEE Transactions on Automatic Control*, vol. 42, no. 9, pp. 1294–1299, Sep. 1997.

[14] M. Yu, L. Wang, T. Chu, and G. Xie, “Stabilization of networked control systems with data packet dropout and network delays via switching system approach,” in *Proceedings of 43rd IEEE Conference on Decision and Control*, vol. 4, pp. 3539–3544, Dec. 2004.

[15] M. Yu, L. Wang, T. Chu, and F. Hao, “An LMI approach to networked control systems with data packet dropout and transmission delays,” in *Proceedings of 43rd IEEE Conference on Decision and Control*, vol. 4, pp. 3545–3550, Dec. 2004.

[16] W.-A. Zhang and L. Yu, “Modelling and control of networked control systems with both network-induced delay and packet-dropout,” *Automatica*, vol. 44, no. 12, pp. 3206–3210, Dec. 2008.

[17] J. Xiong and J. Lam, “Stabilization of linear systems over networks with bounded packet loss,” *Automatica*, vol. 43, no. 1, pp. 80–87, Jan. 2007.

[18] M. Sahebsara, T. Chen, and S. L. Shah, “Optimal filtering with random sensor delay, multiple packet dropout and uncertain observations,” *International Journal of Control*, vol. 80, no. 2, pp. 292–301, Feb. 2007.

[19] H. Zhang, M. Li, J. Yang, and D. Yang, “Fuzzy model-based robust networked control for a class of nonlinear systems,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 39, no. 2, pp. 437–447, Mar. 2009.

[20] Z. Wang, F. Yang, D. W. C. Ho, and X. Liu, “Robust control for networked systems with random packet losses,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 4, pp. 916–924, Aug. 2007.

[21] M. Velasco, J. M. Fuertes, C. Lin, P. Martí, and S. Brandt, "A control approach to bandwidth management in networked control systems," in *Proceedings of the 30TH Annual Conference of IEEE Industrial Electronics Society,* vol.3, pp. 2343-2348, Nov.2004.

[22] Z.-H. Guan, C.-Y. Chen, G. Feng, and T. Li, "Optimal tracking performance limitation of networked control systems sith limited bandwidth and additive colored white Gaussian noise," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 60, no. 1, pp. 189–198, Jan. 2013.

[23] R. Castane, P. Marti, M. Velasco, A. Cervin, and D. Henriksson, "Resource management for control tasks based on the transient dynamics of closed-loop systems," in *Proceedings of 18th Euromicro Conference on Real-Time Systems, Dresden, Germany,* pp. 10–20, Jul. 2006.

[24] Y.-C. Lin and F.-L. Lian, "Data reduction and bandwidth allocation for video-based network system," in *Proceedings of 2012 International Conference on Information and Automation,* pp. 116–121, Jun. 2012.

[25] P. Belzarena, A. Ferragut, and F. Paganini, "Network bandwidth allocation via distributed auctions with time reservations," in *Proceedings of IEEE INFOCOM 2009,* Rio de Janeiro, Brazil, pp. 2816–2820, Apr. 2009.

[26] M. S. Branicky, S. M. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proceedings of the 41st IEEE Conference on Decision and Control,* vol. 2, pp. 1211–1217, Dec. 2002.

[27] G. Weiss, S. Fischmeister, M. Anand, and R. Alur, “Specification and analysis of network resource requirements of control systems,” in *Proceedings of Hybrid Systems: Computation and Control*, R. Majumdar and P. Tabuada, Springer, Berlin, Heidelberg, pp. 381–395, Apr. 2009.

[28] L. Xu, M. Fei, T. Jia, and T. C. Yang, “Bandwidth scheduling and optimization using non-cooperative game model-based shuffled frog leaping algorithm in a networked learning control system,” *Neural Computing & Applications*, vol. 21, no. 6, pp. 1117–1128, Sep. 2011.

[29] F.-L. Lian, J. Moyne, and D. Tilbury, “Time delay modeling and sample time selection for networked control systems,” in *Proceedings of International Mechanical Engineering Congress and Exposition*, New York, vol. 20, pp. 5–6, Nov. 2001.

[30] F.-L. Lian, J. Moyne, and D. Tilbury, “Network design consideration for distributed control systems,” *IEEE Transactions on Control Systems Technology*, vol. 10, no. 2, pp. 297–307, Mar. 2002.

[31] J. Dong and W.-J. Kim, “Bandwidth allocation of networked control systems with exponential approximation,” in *Proceedings of the ASME 2013 Dynamic Systems and Control Conference*, vol. 2, pp. V002T20A002, Oct. 2013.

[32] K. Kim and W.-J. Kim, “Performance-index functions in networked control systems with disturbance and noise,” in *Proceedings of the ASME 2015 International Mechanical Engineering Congress and Exposition*, Vol. 4B, pp. V04BT04A020, Nov. 2015.

[33] K. J. Hunt, D. Sbarbaro, R. Żbikowski, and P. J. Gawthrop, “Neural networks for control systems—A survey,” *Automatica*, vol. 28, no. 6, pp. 1083–1112, Nov. 1992.

[34] J. Yi, Q. Wang, D. Zhao, and J. T. Wen, “BP neural network prediction-based variable-period sampling approach for networked control systems,” *Applied Mathematics and Computation*, vol. 185, no. 2, pp. 976–988, Feb. 2007.

[35] G. Wiederhold and J. McCarthy, “Arthur Samuel: Pioneer in machine learning,” *IBM Journal of Research and Development*, vol. 36, no. 3, pp. 329–331, May 1992.

[36] M. Zastrow, “Machine outsmarts man in battle of the decade,” *New Scientist*, vol. 229, no. 3065, p. 21, Mar. 2016.

[37] M. Dorigo and U. Schnepf, “Genetics-based machine learning and behavior-based robotics: A new synthesis,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 1, pp. 141–154, Jan. 1993.

[38] J. Nagumo and A. Noda, “A learning method for system identification,” *IEEE Transactions on Automatic Control*, vol. 12, no. 3, pp. 282–287, Jun. 1967.

[39] B. K. Panigrahi and V. R. Pandi, “Optimal feature selection for classification of power quality disturbances using wavelet packet-based fuzzy k-nearest neighbour algorithm,” *IET Transactions on Generation*, vol. 3, no. 3, pp. 296–306, Mar. 2009.

[40] M.-S. Tsai, M.-T. Lin, and H.-T. Yau, “Development of command-based iterative learning control algorithm with consideration of friction, disturbance, and noise

effects," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 3, pp. 511–518, May 2006.

[41] M. Mantere, I. Uusitalo, M. Sailio, and S. Noponen, "Challenges of machine learning based monitoring for industrial control system networks," in *Proceedings of the 26th International Conference on Advanced Information Networking and Applications Workshops*, pp. 968–972, Mar. 2012.

[42] M. Mantere, M. Sailio, and S. Noponen, "Feature selection for machine learning based anomaly detection in industrial control system networks," in Proceedings of IEEE International Conference on Green Computing and Communications, pp. 771–774, Nov. 2012.

[43] J. S. Jang, Y. L. Kim, and J. H. Park, "A study on the optimization of the uplink period using machine learning in the future IoT network," in *Proceedings of 2016 IEEE International Conference on Pervasive Computing and Communication Workshops*, pp. 1–3, Mar. 2016.

[44] X. Tian, X. Wang, and Y. Cheng, "A Self-tuning fuzzy controller for networked control system," *International Journal of Computer Science and Network Security*, vol. 7, no. 1, pp. 97–102, Jan. 2007.

[45] G. C. Walsh and H. Ye, "Scheduling of networked control systems," *IEEE Control Systems*, vol. 21, no. 1, pp. 57–65, Feb. 2001.

[46] J. H. Braslavsky, R. H. Middleton, and J. S. Freudenberg, "Feedback stabilization over signal-to-noise ratio constrained channels," in *Proceedings of 2004 American Control Conference*, Boston, MA, vol. 6, pp. 4903–4908, Jul. 2004.

[47] N. Elia and S. K. Mitter, “Stabilization of linear systems with limited information,” *IEEE Transactions on Automatic Control*, vol. 46, no. 9, pp. 1384–1400, Sep. 2001.

[48] G. N. Nair and R. J. Evans, “Exponential stabilisability of finite-dimensional linear systems with limited data rates,” *Automatica*, vol. 39, no. 4, pp. 585–593, Apr. 2003.

[49] S. Tatikonda and S. Mitter, “Control under communication constraints,” *IEEE Transactions on Automatic Control*, vol. 49, no. 7, pp. 1056–1068, Jul. 2004.

[50] D. Liberzon and J. P. Hespanha, “Stabilization of nonlinear systems with limited information feedback,” *IEEE Transactions on Automatic Control*, vol. 50, no. 6, pp. 910–915, Jun. 2005.

[51] G. N. Nair, R. J. Evans, I. M. Y. Mareels, and W. Moran, “Topological feedback entropy and nonlinear stabilization,” *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1585–1597, Sep. 2004.

[52] M. Y. Chow and Y. Tipsuwan, “Time sensitive network-based control systems and applications,” *IEEE IES Network Based Control Newsletter*, vol. 5, no. 2, pp. 13–18, 2005.

[53] W. Fei-Yue and L. Derong, *Networked Control Systems: Theory and Applications*. Springer, London, 2008.

[54] W. Zhang, M. S. Branicky, and S. M. Phillips, “Stability of networked control systems,” *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 84–99, Feb. 2001.

[55] F.-L. Lian, J. R. Moyne, and D. M. Tilbury, “Performance evaluation of control networks: Ethernet, ControlNet, and DeviceNet,” *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 66–83, Feb. 2001.

[56] V. Tuzlukov, *Signal processing noise*, 1st Edition. CRC Press, 2002.

[57] R. Mancini and B. Carter, *Op amps for everyone*. Texas Instruments, 2002.

[58] D. Guo, S. Shamai, and S. Verdu, “Mutual information and minimum mean-square error in Gaussian channels,” *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1261–1282, Apr. 2005.

[59] A. Klein, G. K. Kaleh, and P. W. Baier, “Zero forcing and minimum mean-square-error equalization for multiuser detection in code-division multiple-access channels,” *IEEE Transactions on Vehicular Technology*, vol. 45, no. 2, pp. 276–287, May 1996.

[60] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[61] C. J. C. H. Watkins, *Learning from Delayed Rewards*, University of Cambridge, England, 1989.

[62] K. Ji, “Real-time control over networks (Doctoral dissertation),” Texas A&M University, 2003.

[63] “Samba.” [Online]. Available: https://www.samba.org/. [Accessed: 30-Jan-2019].

[64] S. C. Paschall II, “Design, fabrication, and control of a single actuator magnetic levitation system (Senior honors thesis),” Texas A&M University, 2002.

[65] M. Lee, “Real-time networked control with multiple clients (Master’s thesis),” Texas A&M University, 2009.

[66] P.-C. Hsieh, “Autonomous robotic wheelchair with collision-avoidance navigation (Master’s thesis),” Texas A&M University, 2008.

[67] K. Ji and W.-J. Kim, “Optimal bandwidth allocation and QoS-adaptive control co-design for networked control systems,” *International Journal of Control, Automation, and Systems*, vol. 6, no. 4, pp. 596–606, Aug. 2008.

[68] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer, 1999.

[69] J. Dong, “Output feedback control and optimal bandwidth allocation of networked control systems (Doctoral dissertation),” Texas A&M University, 2013.

APPENDIX A

C/C++ CODES FOR THE NCS

A.1 C Code for Server [69]

```
//This code includes all the control laws for six clients: a ball maglev, four DC motors, //
and an Autonomous wheelchair.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <asm/errno.h>
#include <sys/types.h>
#include <sys/user.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sched.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <errno.h>
#include <inttypes.h>
#include "defines.h"
```

```
#define KEEP_STATIC_INLINE
#include <rtai_lxrt_user.h>
#include <rtai_lxrt.h>
RTIME time_stamp;
//data from server to client
double u0; //current control data
double u1e; //below are predicted control data for ball maglev
double u2e;
double u3e;
double u4e;
double u5e;
double u6e;
double u7e;
double u8e;
//data from client to server
double y0; //below are sensor data from clients
double y_1;
double y_2;
double y_3;
double y_4;
double y_5;
double y_6;
double y_7; //client ID number.
double u_1; //control data
double u_2; //control data
//AR model predicted data
double y_hat_1;
double y_hat_2;
double y_hat_3;
```

```
double y_hat_4;
double y_hat_5;
double y_hat_6;
double y_hat_7;
double y_hat_8;
FILE *fp;

// variables for noise
FILE *fp1;
float noise[20000];
int idx=0;
float dist=0;    // multiplier for disturbance

//variables for DC motor 1
double y_dot_desi = 7.0; //desire speed of DC motor [rps]
double e0 = 0;
double e1 = 0;
double u0 = 0;
double u1 = 0;
float e_dist = 0; // estimated disturbance

//DC motor 2
double y_dot_desi2 = 7.0;

//DC motor 3
double y_dot_desi3 = 7.0;

//DC motor 4
double y_dot_desi4 = 7.0;
```

```
// Variables for Ball Maglev
double y_hat_desi = 0.005; // User input (desired set point)
double v_hat_err = 0.0;
double k = 0.083; //small k -> small vibration current setting (Gain parameter)
double c = 0.0; //Controller constant
double v = 0.975; //current setting for initial offset
double er0 = 0.0; //Input to controller at time n
double er1 = 0.0; //Intput to controller at time n-1
double er2 = 0.0; //Intput to controller at time n-2
int i=0;

// for display, file print counters
int p1 = 0;
int p2 = 0;
int p3 = 0;
int p4 = 0;
int p5 = 0;
int p6 = 0;

//rtai declarations
unsigned long mtsk_name;
RT_TASK *mtsk;
struct sched_param mysched;
void terminate_normally(int signo)
{
  fflush(stdin);
        if(signo==SIGINT || signo==SIGTERM)
        {
                printf("Terminating the program normally\n");
```

```
                //make the process soft real time process
                rt_make_soft_real_time();
                printf("MASTER TASK YIELDS ITSELF\n");
                rt_task_yield();
                printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
                stop_rt_timer();
                printf("MASTER TASK DELETES ITSELF\n");
                rt_task_delete(mtsk);
                printf("END MASTER TASK\n");
        }
  exit(0);
}

main(int argc, char *argv[])
{
        int sockid, nread, addrlen;
        int nw, nr;
        int send_buffer_size, recv_buffer_size;
        unsigned short server_port = 0;
        struct sockaddr_in my_addr, client_addr;
        struct send_data *send_buffer = NULL;
        struct recv_data *recv_buffer = NULL;
        RTIME start_time = 0;
        RTIME end_time = 0;
        RTIME actual_period = 0;
        RTIME difference = 0;
        size_t iRet = 0;
        int esti_count = 0;
        double vhaterr_prev[5] = {0.0, 0.0, 0.0, 0.0, 0.0};
```

```
int j=0;
//signal handling
struct sigaction sa;
//Initialize the signal handling structure
sa.sa_handler = terminate_normally;
sa.sa_flags = 0;
sigemptyset(&sa.sa_mask);
if(sigaction(SIGINT, &sa, NULL))
{
        perror("sigaction");
}
if(sigaction(SIGTERM, &sa, NULL))
{
        perror("sigaction");
}
fprintf(stderr, "creating socket\n");
if ( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
perror("socket() failed ");
fprintf(stderr, "%s: socket error: %d\n", argv[0], errno);
exit(2);
}
fprintf(stderr, "binding my local socket\n");
server_port = 4444;
memset((void *) &my_addr, (char) 0, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htons(INADDR_ANY);
my_addr.sin_port = htons(server_port);
if ( (bind(sockid, (struct sockaddr *) &my_addr, sizeof(my_addr)) < 0) ){
perror("bind() failed ");
```

```
        fprintf(stderr, "bind() errno = %d\n", errno);
        exit(4);
        }
        recv_buffer_size = sizeof(struct recv_data);
        if(( recv_buffer = (struct recv_data *)calloc(1, sizeof(struct recv_data)))
==NULL) {
                fprintf(stderr, "cannot allocate memory for buffer!\n");
                exit(4);
        }
        send_buffer_size = sizeof(struct send_data);
        if(( send_buffer = (struct send_data *)calloc(1, sizeof(struct send_data)))
==NULL) {
                fprintf(stderr, "cannot allocate memory for buffer!\n");
                exit(4);
        }
        addrlen = sizeof(client_addr);
        fprintf(stderr, "%s: starting blocking message read\n", argv[0]);
        mysched.sched_priority = 99;
        if( sched_setscheduler( 0, SCHED_FIFO, &mysched ) == -1 ) {
        puts(" ERROR IN SETTING THE SCHEDULER UP");
        perror( "errno" );
        exit( 0 );
        }
        mlockall(MCL_CURRENT | MCL_FUTURE);
        mtsk_name = nam2num("MTSK");
        if (!(mtsk = rt_task_init(mtsk_name, 0, 0, 0))) {
                printf("CANNOT INIT MASTER TASK\n");
                exit(1);
        }
```

```
        start_time = rt_get_cpu_time_ns();
        printf("main: start_time = %lld\n", start_time);
        printf("MASTER TASK STARTS THE ONESHOT TIMER\n");
        actual_period = start_rt_timer(nano2count(25000));
        printf("actual_period = %lld\n", actual_period);
        printf("MASTER TASK MAKES ITSELF PERIODIC \n");
        rt_task_make_periodic(mtsk, rt_get_time()+ nano2count(3000000),
nano2count(3000000));

        fp = open("result.txt", "w"); // Save result
        fp1 = fopen("random.txt", "r"); // Read white gaussian noise

        if(fp1 == NULL)
        {
           printf("could not open file.\n");
        }
        while(!feof(fp1)) {
        fscanf(fp1, "%f,", &noise[idx]);
        idx+=1;
        }

while( 1 )
{
        FILE *fp = NULL;
        float print_data[200000];
        int loop_count = 0;
        fp = fopen("result.txt", "W");
        if(fp == NULL)
        {
```

```
            printf("could not open file");
        }
        nr = recvfrom(sockid, (void *)recv_buffer, recv_buffer_size, 0, (struct sockaddr
*) &client_addr, &addrlen);
        if( nr <= -1 ) {
                fprintf(stderr, "recvfrom() errno = %d\n", errno);
                exit(10);
        }
        start_time = rt_get_cpu_time_ns();
        y0 = recv_buffer->y0;
        y_1 = recv_buffer->y_1;
        y_2 = recv_buffer->y_2;
        y_3 = recv_buffer->y_3;
        y_4 = recv_buffer->y_4;
        y_5 = recv_buffer->y_5;
        y_6 = recv_buffer->y_6;
        y_7 = recv_buffer->y_7;
        u_1 = recv_buffer->u_1;
        u_2 = recv_buffer->u_2;
        e_dist = u_1; // estimated disturbance calculated from client

        if(y_7 == 1) {  //Client 1 Ball Maglev
                er0 = (y_hat_desi - (-0.0010108*y0+0.0114970))*k;  // Error Calculation
                er1 = (y_hat_desi - (-0.0010108*y_1+0.0114970))*k;
                er2 = (y_hat_desi - (-0.0010108*y_2+0.0114970))*k;
                u0 = ((0.782*(u_1-v)) + (0.13*(u_2-v)) + (41500.0*er0) -
(41500.0*1.754*er1) + (41500.0*0.769*er2)) + v;
                send_buffer->u0 = u0;
```

```
y_hat_1 = 0.8122*y0 - 0.3479*y_1 - 0.0294*y_2 + 0.4605*y_3 + 0.0742*y_4 +
0.1042*y_5 + 0.1117*y_6;// - 0.3561*y_7;
er0 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;  // Error Calculation
er1 = (y_hat_desi - (-0.0010108*y0+0.0114970))*k;
er2 = (y_hat_desi - (-0.0010108*y_1+0.0114970))*k;
u1e = ((0.782*(u0-v)) + (0.13*(u_1-v)) + (41500.0*er0) - (41500.0*1.754*er1) +
(41500.0*0.769*er2)) + v;
send_buffer->u1e = u1e;
y_hat_2 = 0.3117*y0 - 0.3119*y_1 + 0.4366*y_2 + 0.4482*y_3 + 0.1645*y_4 +
0.1964*y_5 - 0.2653*y_6;// - 0.2892*y_7;
er0 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;  / Error Calculation
er1 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;
er2 = (y_hat_desi - (-0.0010108*y0+0.0114970))*k;
u2e = ((0.782*(u1e-v)) + (0.13*(u0-v)) + (41500.0*er0) - (41500.0*1.754*er1) +
(41500.0*0.769*er2)) + v;
send_buffer->u2e = u2e;
y_hat_3 = -0.0587*y0 + 0.3281*y_1 + 0.4390*y_2 + 0.3080*y_3 + 0.2195*y_4
- 0.2329*y_5 - 0.2544*y_6;// - 0.1110*y_7;
er0 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;  // Error Calculation
er1 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;
er2 = (y_hat_desi - (-0.0010108*y_hat_1+0.0114970))*k;
u3e = ((0.782*(u2e-v)) + (0.13*(u1e-v)) + (41500.0*er0) - (41500.0*1.754*er1)
+ (41500.0*0.769*er2)) + v;
send_buffer->u3e = u3e;
y_hat_4 = 0.2804*y0 + 0.4594*y_1 + 0.3097*y_2 + 0.1925*y_3 - 0.2372*y_4 -
0.2605*y_5 - 0.1176*y_6;// + 0.0209*y_7;
er0 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;  // Error Calculation
er1 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;
er2 = (y_hat_desi - (-0.0010108*y_hat_2+0.0114970))*k;
```

```
        u4e = ((0.782*(u3e-v)) + (0.13*(u2e-v)) + (41500.0*er0) - (41500.0*1.754*er1)
+ (41500.0*0.769*er2)) + v;
        send_buffer->u4e = u4e;
        y_hat_5 = 0.6872*y0 + 0.2122*y_1 + 0.1842*y_2 - 0.1081*y_3 - 0.2397*y_4 -
0.0884*y_5 + 0.0523*y_6;// - 0.0999*y_7;
        er0 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;  / Error Calculation
        er1 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;
        er2 = (y_hat_desi - (-0.0010108*y_hat_3+0.0114970))*k;
        u5e = ((0.782*(u4e-v)) + (0.13*(u3e-v)) + (41500.0*er0) - (41500.0*1.754*er1)
+ (41500.0*0.769*er2)) + v;
        send_buffer->u5e = u5e;
        y_hat_6 = 0.7703*y0 - 0.0548*y_1 - 0.1283*y_2 + 0.0767*y_3 - 0.0374*y_4 +
0.1239*y_5 - 0.0231*y_6;// - 0.2447*y_7;
        er0 = (y_hat_desi - (-0.0010108*y_hat_6+0.0114970))*k;  // Error Calculation
        er1 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;
        er2 = (y_hat_desi - (-0.0010108*y_hat_4+0.0114970))*k;
        u6e = ((0.782*(u5e-v)) + (0.13*(u4e-v)) + (41500.0*er0) - (41500.0*1.754*er1)
+ (41500.0*0.769*er2)) + v;
        send_buffer->u6e = u6e;
        y_hat_7 = 0.5708*y0 - 0.3963*y_1 + 0.0541*y_2 + 0.3173*y_3 + 0.1810*y_4 +
0.0572*y_5 - 0.1586*y_6;// - 0.2743*y_7;
        er0 = (y_hat_desi - (-0.0010108*y_hat_7+0.0114970))*k;  // Error Calculation
        er1 = (y_hat_desi - (-0.0010108*y_hat_6+0.0114970))*k;
        er2 = (y_hat_desi - (-0.0010108*y_hat_5+0.0114970))*k;
        u7e = ((0.782*(u6e-v)) + (0.13*(u5e-v)) + (41500.0*er0) - (41500.0*1.754*er1)
+ (41500.0*0.769*er2)) + v;
        send_buffer->u7e = u7e;
        send_buffer->u8e = 1;
```

```
p1 = p1+1;
fprintf(fp,"m02 %d\n",p1);
printf("m02  %d\n",p1);
}

if(y_7 == 2){  //Client 2 DC motor
        e1 = y_dot_desi - y0; // Error Calculation

//control law u(k)=u(k-1)+(1.5+2.5h)*e(k)+(2.5h-1.5)*e(k-1)
//PI controller with Kp=1.5 Ki=5
u0 = u1 + (1.8+3*h1)*e1 - (1.8-3*h1)*e0 + noise[p2]*0.267;   //* universal
digital controller (Tustin's method)
send_buffer->u0 = u0;
send_buffer->u1e = u1e;
u2e = 0;
send_buffer->u2e = u2e;
u3e = 0;
send_buffer->u3e = u3e;
u4e = 0;
send_buffer->u4e = u4e;
u5e = 0;
send_buffer->u5e = u5e;
u6e = 0;
send_buffer->u6e = u6e;
u7e = 0;
send_buffer->u7e = u7e;
u8e = 2;
send_buffer->u8e = u8e;
```

```
p2 = p2+1;
printf("          DC1 %d\n", p2);
fprintf(fp,"                    DC1  %d\n",p2);
u1 = u0;
u2 = u1;
e0 = e1;
}

if(y_7 == 3) { //Client 3 wheelchair robot
        if(strcmp(y_8,"00010")==0){
                send_buffer->u0 = 2.0;
        }
        else if(strcmp(y_8,"00011")==0){
                send_buffer->u0 = 2.0;
        }
        else if(strcmp(y_8,"00100")==0){
                send_buffer->u0 = 2.0;
        }
        else if(strcmp(y_8,"00101")==0){
                send_buffer->u0 = 2.0;
        }
        else if(strcmp(y_8,"00110")==0){
                send_buffer->u0 = 2.0;
        }
        else if(strcmp(y_8,"00111")==0){
                send_buffer->u0 = 2.0;
        }
        else if(strcmp(y_8,"01000")==0){
                send_buffer->u0 = 7.0;
```

```
}
else if(strcmp(y_8,"01001")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"01010")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"01100")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"01110")==0){
        send_buffer->u0 = 2.0;
}
else if(strcmp(y_8,"10000")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"10001")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"10010")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"10100")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"10110")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"11000")==0){
```

```
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"11001")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"11010")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"11100")==0){
        send_buffer->u0 = 7.0;
}
else if(strcmp(y_8,"11110")==0){
        send_buffer->u0 = 10.0;
}
else if(strcmp(y_8,"00000")==0){
        send_buffer->u0 = 10.0;
}
else{
        send_buffer->u0 = 5.0;
}
u1e = 0;
send_buffer->u1e = u1e;
u2e = 0;
send_buffer->u2e = u2e;
u3e = 0;
send_buffer->u3e = u3e;
u4e = 0;
send_buffer->u4e = u4e;
u5e = 0;
```

```
        send_buffer->u5e = u5e;
        u6e = 0;
        send_buffer->u6e = u6e;
        u7e = 0;
        send_buffer->u7e = u7e;
        u8e = 2;
        send_buffer->u8e = u8e;
        p3 = p3+1;
        printf("                                    _wheelchair %d_\n", p3);
        fprintf(fp,"                                        wheelchair %d\n",p3);
        u1 = u0;
        e0 = e1;
}

if(y_7 == 4) { //Client 4 DC motor2
        e1 = y_dot_desi2 - y0;  // Error Calculation
        u0 = u1 + (1.8+3*h2)*e1 - (1.8-3*h2)*e0 + noise[p4]*0.267;   // universal
digital controller (Tustin's method)

        send_buffer->u0 = u0;
        u1e = 0;
        send_buffer->u1e = u1e;
        u2e = 0;
        send_buffer->u2e = u2e;
        u3e = 0;
        send_buffer->u3e = u3e;
        u4e = 0;
        send_buffer->u4e = u4e;
        u5e = 0;
```

```
                send_buffer->u5e = u5e;
                u6e = 0;
                send_buffer->u6e = u6e;
                u7e = 0;
                send_buffer->u7e = u7e;
                u8e = 2;
                send_buffer->u8e = u8e;
                p4 = p4+1;
                printf("                                DC2 %d\n", p4);
                fprintf(fp,"                                    DC2  %d\n",p4);
                u1 = u0;
                e0 = e1;
        }

        if(y_7 == 5) { //Client 5 DC motor3
                u0 = u1 + (1.8+3*h3)*e1 - (1.8-3*h3)*e0 + noise[p5]*0.267;   // universal
digital controller (Tustin's method)

                send_buffer->u0 = u0;
                u1e = 0;
                send_buffer->u1e = u1e;
                u2e = 0;
                send_buffer->u2e = u2e;
                u3e = 0;
                send_buffer->u3e = u3e;
                u4e = 0;
                send_buffer->u4e = u4e;
                u5e = 0;
                send_buffer->u5e = u5e;
```

```
                u6e = 0;
                send_buffer->u6e = u6e;
                u7e = 0;
                send_buffer->u7e = u7e;
                u8e = 2;
                send_buffer->u8e = u8e;
                p5 = p5+1;
                printf("                                        DC3 %d\n", p5);
                fprintf(fp,"                                            DC3  %d\n",p5);
                u1 = u0;
                e0 = e1;
        }

        if(y_7 == 6) { //Client 6 DC motor4
                e1 = y_dot_desi4 - y0;  // Error Calculation
                u0 = u1 + (1.8+3*h4)*e1 - (1.8-3*h4)*e0 + noise[p6]*0.267;   // universal
digital controller (Tustin's method)

                send_buffer->u0 = u0;
                u1e = 0;
                send_buffer->u1e = u1e;
                u2e = 0;
                send_buffer->u2e = u2e;
                u3e = 0;
                send_buffer->u3e = u3e;
                u4e = 0;
                send_buffer->u4e = u4e;
                u5e = 0;
                send_buffer->u5e = u5e;
```

```
                u6e = 0;
                send_buffer->u6e = u6e;
                u7e = 0;
                send_buffer->u7e = u7e;
                u8e = 2;
                send_buffer->u8e = u8e;
                p6 = p6+1;
                printf("                                        DC4 %d\n", p6);
                fprintf(fp,"                                      DC4  %d\n",p6);
                u1 = u0;
                e0 = e1;
        }

        end_time = rt_get_cpu_time_ns();
        send_buffer->time_stamp = recv_buffer->time_stamp;
        nw = sendto(sockid, (const void *)send_buffer, send_buffer_size, 0, (struct
sockaddr *) &client_addr, addrlen);

        if( nw <= -1 ) {
                perror("sendto failed ");
                fprintf(stderr, "sendto() errno = %d \n", errno);
                exit(12);
        }
} // END of while
fclose(fp);
fclose(fp1);

printf("MASTER TASK YIELDS ITSELF\n");
rt_task_yield();
```

```
printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
stop_rt_timer();
printf("MASTER TASK DELETES ITSELF\n");
rt_task_delete(mtsk);
close(sockid);
free(send_buffer);
free(recv_buffer);
}
```

A.2 C Code for Client (Ball Maglev System) [69]

```
// A Ball Maglev system
#include <stdio.h>

…       // Here is an identical code block as the one in Appendix A.1

#define PERIOD 1000000
#define LOOPS 1000
#define NTASKS 2
#define taskname(x) (1000 + (x))

…       // Here is an identical code block as the one in Appendix A.1

double y_7=1;
double u_1;
double u_2;
RTIME current_time_stamp;
pthread_t task[NTASKS];
int ntasks = NTASKS;
```

```
RT_TASK *mytask;
SEM *sem;
static int cpus_allowed;
SEM *sock_sem; //socket semaphoere, used by all the threads.
int sockid;
RTIME start_instant;
int server_sock_size = 0;
struct sockaddr_in my_addr, server_addr;
comedi_t *it;
int in_subdev = 0; // digital input
int out_subdev = 1; // analog output
int in_chan = 0;
int out_chan = 0;
int in_range = 0;
int out_range = 0;
int aref = AREF_GROUND;
int i=0;

//comedi declarations
lsampl_t in_data;
lsampl_t out_data;
float volts = 0.0;
int in_maxdata = 0, out_maxdata = 0;
comedi_range *in_range_ptr, *out_range_ptr;
int endme_int = 0;

void terminate_normally(int signo);
void endme(int sig)
{
```

```
        printf("You want to kill me?\n");
        endme_int = 1;
        rt_sem_delete(sem);
        comedi_close(it);
        stop_rt_timer();
        rt_task_delete(mytask);
        signal(SIGINT, SIG_DFL);
        exit(1);
}

void *send_thread_fun(void *arg)
{
        RTIME start_time, period, end_time, difference;
        RTIME t0;
        SEM *sem;
        RT_TASK *mytask;
        unsigned long mytask_name;
        int mytask_indx;
        double * buffer = NULL;
        int iRet = 0;
        struct recv_data *send_msg = NULL;
        int send_msg_size;
        FILE *fp = NULL;
        float print_data[20000];
        int loop_count = 0;
        fp = fopen("result.txt","w");
        if(fp == NULL)
        {
                printf("could not open file");
```

```
                exit(0);
        }
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
        mytask_indx = 0;
        mytask_name = taskname(mytask_indx);
        cpus_allowed = 1 - cpus_allowed;
        if (!(mytask = rt_task_init_schmod(mytask_name, 1, 0, 0, SCHED_FIFO, 1 <<
cpus_allowed))) {
                printf("CANNOT INIT send_thread TASK\n");
                exit(1);
        }
        printf("send thread pid = %ld\t master pid = %ld\n", getpid(), getppid());
        mlockall(MCL_CURRENT | MCL_FUTURE);
        rt_receive(0, (unsigned int*)&sem);
        send_msg_size = sizeof(struct recv_data);
        if(( send_msg = (struct recv_data *)calloc(1, sizeof(struct recv_data))) ==
NULL)
        {
        printf("cannot allocate message memory\n");
                exit(4);
        }

        period = nano2count(PERIOD);
        start_time = rt_get_time() + nano2count(10000000);
        t0 = start_instant;
        printf("send: t0 = %lld\t", t0);
        printf("This period = %lld\t", rt_get_time());
        printf("actual start = %lld\n", t0 + nano2count(500000000));
```

```
	rt_task_make_periodic(mytask, (t0 + nano2count(500000000)),
nano2count(3000000));//3.0 ms sampling time
	start_time = rt_get_cpu_time_ns();
	printf("starting the send_thread while loop\n");
	for(;;) {
		if(endme_int == 1) {
			break;
		}

		comedi_data_read(it, in_subdev, in_chan, in_range, aref, &in_data);
		if(in_data > 4094) {
			in_data = 4094;
		}
		if(in_data < 2049) {
			in_data = 2049;
		}
		current_time_stamp = rt_get_cpu_time_ns();
		printf("in_data: %i",in_data);
		printf("in_range_ptr: %i",in_range_ptr);
		printf("in_maxdata: %i",in_maxdata);
		y_0 = comedi_to_phys(in_data, in_range_ptr, in_maxdata);
		print_data[loop_count] = u_1; //output control data
		printf("y_0 : %f \n",y_0);
		send_msg->y_0 = y_0;
		send_msg->y_1 = y_1;
		send_msg->y_2 = y_2;
		send_msg->y_3 = y_3;
		send_msg->y_4 = y_4;
		send_msg->y_5 = y_5;
```

```
                send_msg->y_6 = y_6;
                send_msg->y_7 = y_7;
                send_msg->u_1 = u_1;
                send_msg->u_2 = u_2;
                send_msg->time_stamp = current_time_stamp;

                rt_sem_wait(sock_sem);
                iRet = sendto(sockid, (const void *)send_msg, send_msg_size, 0, (struct
sockaddr*)&server_addr, server_sock_size);
                rt_sem_signal(sock_sem);
                if(iRet <= -1) {
                        perror("sendto() failed\n");
                        break;
                }
                //y_7 = y_6;
                y_6 = y_5;
                y_5 = y_4;
                y_4 = y_3;
                y_3 = y_2;
                y_2 = y_1;
                y_1 = y_0;
                loop_count++;
                if(loop_count == 20000) {
                        break;
                }
                rt_task_wait_period();

        }
        end_time = rt_get_cpu_time_ns();
```

```
        difference = end_time - start_time;
        printf("difference = %lld\n", difference);
        endme_int++;
        rt_sem_signal(sem);
        rt_make_soft_real_time();
        for(i=0;i<20000;i++) {
                fprintf(fp, "%f\n", print_data[i]);
        }
        fclose(fp);
        free(send_msg);
        rt_task_delete(mytask);
        printf("send_thread ENDS\n");
        return 0;
} // End of send thread

void *recv_thread_fun(void *arg)
{
        RTIME start_time, period, end_time, difference;
        RTIME t0;
        SEM *sem;
        RT_TASK *mytask;
        unsigned long mytask_name;
        int mytask_indx;
        struct data *buffer = NULL;
        int iRet = 0;
        int recv_msg_size;
        struct send_data *recv_msg = NULL;
        int loop_count = 0;
        recv_msg_size = sizeof(struct send_data);
```

```
	if(( recv_msg = (struct send_data *)calloc(1, sizeof(struct send_data))) ==
NULL) {
	printf("cannot allocate message memory\n");
		exit(4);
	}
	pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
	mytask_indx = 1;
	mytask_name = taskname(mytask_indx);
	cpus_allowed = 1 - cpus_allowed;
	if (!(mytask = rt_task_init_schmod(mytask_name, 1, 0, 0, SCHED_FIFO, 1 <<
cpus_allowed))) {
		printf("CANNOT INIT recv_thread TASK\n");
		exit(1);
	}
	printf("recv thread pid = %ld\t master pid = %ld\n", getpid(), getppid());
	mlockall(MCL_CURRENT | MCL_FUTURE);
	rt_receive(0, (unsigned int*)&sem);
	period = nano2count(PERIOD);
	start_time = rt_get_time() + nano2count(10000000);
	t0 = start_instant;
	printf("recv: t0 = %lld\t", count2nano(t0));
	printf("This period = %lld\t", count2nano(rt_get_time()));
	printf("actual start = %lld\n", count2nano(t0 + nano2count(500500000)));
	rt_task_make_periodic(mytask, (t0 + nano2count(500500000)),
nano2count(3000000));//smapling period 3ms
	start_time = rt_get_time();
	printf("starting the recv_thread while loop\n");
	for(;;) {
		if(endme_int == 1) {
```

```
                break;
        }
        rt_sem_wait(sock_sem);
        iRet = recvfrom(sockid, (void *)recv_msg, recv_msg_size, 0, (struct
sockaddr *)&server_addr, &server_sock_size);
        rt_sem_signal(sock_sem);
        if(iRet <= -1) {
                endme_int = 1;
                perror("recvfrom() failed\n");
                break;
        }
        if((loop_count < 30001)) {
        u0 = recv_msg->u0;
        printf("u0 : %f \n",u0);
        u1e = recv_msg->u1e;
        u2e = recv_msg->u2e;
        u3e = recv_msg->u3e;
        u4e = recv_msg->u4e;
        u5e = recv_msg->u5e;
        u6e = recv_msg->u6e;
        u7e = recv_msg->u7e;
        u8e = recv_msg->u8e;
        }

    if(u8e == 1) {
        if(loop_count < 30001) {
                volts = u0;
        }
        else if(loop_count == 30001) { //every 9 seconds data missing
```

```
                volts = u4e;
        }
        if(volts > 10.0) {
                volts = 9.99999;
        }
        if(volts < 0.0) {
                volts = 0.0;
        }
        out_data = comedi_from_phys(volts, out_range_ptr, out_maxdata);
        comedi_data_write(it, out_subdev, out_chan, out_range, aref, out_data);
        u_2 = u_1;
        u_1 = u0;
        if(loop_count == 30001) {
                loop_count = 0;
        }
        Else {
                loop_count++;
        }
}
else
{
        printf("wait\n");
}
rt_task_wait_period();
}
end_time = rt_get_cpu_time_ns();
difference = end_time - start_time;
//printf("difference = %lld\n", difference);
endme_int++;
```

```
        rt_make_soft_real_time();
        free(recv_msg);
        rt_task_delete(mytask);
        printf("recv_thread ENDS\n");
        return 0;
} // End of recv thread
int main(void)
{
        int i;
        unsigned long mytask_name = nam2num("MASTER");
        struct sigaction sa;
        char * server_ip = "192.168.138.46";//maglev1
        unsigned short my_port, server_port;
        my_port = 4445;
        server_port = 4444;

        /* -- Create client side socket -- */
        printf("creating socket\n");
        if( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
                perror("socket() failed ");
                exit(2);
        }

        /* -- Initialize client side socket address -- */
        memset((void *) &my_addr, (char) 0, sizeof(my_addr));
        my_addr.sin_family = AF_INET; // Internet Address Family
        my_addr.sin_addr.s_addr = htonl(INADDR_ANY); /* I can receive from any
host */
        my_addr.sin_port = htons(my_port);
```

```
if ( (bind(sockid, (struct sockaddr *) &my_addr, sizeof(my_addr)) < 0) ) {
        perror("bind() failed ");
        exit(3);
}

/* -- Initialize server side socket address -- */
server_sock_size = sizeof(server_addr);
memset((void *) &server_addr, (char) 0, server_sock_size);
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr(server_ip);
server_addr.sin_port = htons(server_port);
sa.sa_handler = endme;
sa.sa_flags = 0;
sigemptyset(&sa.sa_mask);
if(sigaction(SIGINT, &sa, NULL))    {
        perror("sigaction");
}
if(sigaction(SIGTERM, &sa, NULL)) {
        perror("sigaction");
}
it = comedi_open("/dev/comedi0");
if(it == NULL) {
        printf("Could not open comedi\n");
        exit(1);
}
in_maxdata = comedi_get_maxdata(it, in_subdev, in_chan);
out_maxdata = comedi_get_maxdata(it, out_subdev, out_chan);
in_range_ptr = comedi_get_range(it, in_subdev, in_chan, in_range);
out_range_ptr = comedi_get_range(it, out_subdev, out_chan, out_range);
```

```
if (!(mytask = rt_task_init(mytask_name, 1, 0, 0))) {
        printf("CANNOT INIT main TASK \n");
        exit(1);
}
printf("MASTER INIT: name = %lu, address = %p.\n", mytask_name, mytask);
sem = rt_sem_init(10000, 0);
sock_sem = rt_sem_init(nam2num("SOCK"), 1);
rt_set_periodic_mode();
start_rt_timer(nano2count(25000));
start_instant = rt_get_time();
printf("main: start_instant = %lld\n", start_instant);
if (pthread_create(&task[0], NULL, send_thread_fun, &start_instant)) {
        printf("ERROR IN CREATING send_thread\n");
        exit(1);
}
if (pthread_create(&task[1], NULL, recv_thread_fun, &start_instant)) {
        printf("ERROR IN CREATING recv_thread\n");
        exit(1);
}
for (i = 0; i < ntasks; i++) {
        while (!rt_get_adr(taskname(i))) {
                rt_sleep(nano2count(20000000));
        }
}
for (i = 0; i < ntasks; i++) {
        rt_send(rt_get_adr(taskname(i)), (unsigned int)sem);
}
printf("Start waiting for sem\n");
while(endme_int == 0) {
```

```
            rt_sem_wait_timed(sem, nano2count(5000000000));
        }
        printf("Stop waiting for sem\n");
        for (i = 0; i < ntasks; i++) {
            while (rt_get_adr(taskname(i))) {
                rt_sleep(nano2count(20000000));
            }
        }
        rt_sem_delete(sem);
        rt_sem_delete(sock_sem);
        stop_rt_timer();
        comedi_close(it);
        rt_task_delete(mytask);
        printf("MASTER %lu %p ENDS\n", mytask_name, mytask);
        for (i = 0; i < ntasks; i++) {
            pthread_join(task[i], NULL);
        }
        return 0;
}
```

A.3 C Code for Client (DC Motor Systems) [69]

```
// client.c for DC motors
#include <stdio.h>

…   // Here is an identical code block as the one in Appendix A.1
…   // Here is an identical code block as the one in Appendix A.2
…   // Here is an identical code block as the one in Appendix A.1

double y_7=2;
```

```
double u_1;
double u_2;
RTIME current_time_stamp;
int j = 0, m = 0, b0 = 0, b1 = 0, cnt = 0, p = 0;
double speed = 0;

…       // Here is an identical code block as the one in Appendix A.2

void *send_thread_fun(void *arg)
{

…       // Here is an identical code block as the one in Appendix A.2

        // DC motor speed measure
        while(1) {
                if(endme_int == 1) {
                        break;
                }
                // Counting encoder pulses
                start_time = rt_get_cpu_time_ns();
                comedi_dio_config(it, in_subdev, in_chan, COMEDI_INPUT);
                for(j=0;j<500;j++) {    //* for loop
                        m = comedi_data_read(it, in_subdev, in_chan, in_range, aref,
&in_data);
                        if(in_data == 1) {        //* high or low?
                                b1 = 1;}
                        else {
                                b1 = 0;}
                        if(b1 != b0) {   //* count turn-over (H to L or L to H)
```

```
                cnt++;}
                b0 = b1;
        }
        end_time = rt_get_cpu_time_ns(); //* End of the FOR loop
        current_time_stamp = rt_get_cpu_time_ns();
        speed = (cnt*0.214)+1.322; //* DI counting w/ j=1000, PCI-6025E
        y_0 = speed;

        print_data[loop_count] = y_0;

…       // Here is an identical code block as the one in Appendix A.2

        cnt = 0;

        if(loop_count < 5) {
                printf("s: %d\n", loop_count);
        }
        if(loop_count > 9995) {
                printf("s: %d\n", loop_count);
        }
        loop_count++;
        if(loop_count == 10000) {
                break;
        }
        rt_task_wait_period();
}// End of while loop

…       // Here is an identical code block as the one in Appendix A.2
```

```
}       // End of send_thread

void *recv_thread_fun(void *arg)
{
        RTIME start_time, period, end_time, difference, re_time_stamp,
current_cpu_time;
        RTIME time_diff[20000];

…       // Here is an identical code block as the one in Appendix A.2

        tdiff = fopen("timediff.txt","w");
        if(tdiff == NULL) {
                printf("could not open file");
                exit(0);
        }

…       // Here is an identical code block as the one in Appendix A.2

                if(loop_count < 10000)
                {
                u0 = recv_msg->u0;
                u1e = recv_msg->u1e;
                u2e = recv_msg->u2e;
                u3e = recv_msg->u3e;
                u4e = recv_msg->u4e;
                u5e = recv_msg->u5e;
                u6e = recv_msg->u6e;
                u7e = recv_msg->u7e;
                u8e = recv_msg->u8e;
```

```
re_time_stamp = recv_msg->time_stamp;
current_cpu_time = rt_get_cpu_time_ns();
time_diff[loop_count] = current_cpu_time-re_time_stamp;
}

if(loop_count < 9998) {
        volts = u0*1.014-0.005;
}
else if(loop_count == 9998) { //every  seconds data missing
        volts = 0.0;
}
if(loop_count < 5) {
        printf("r: %d\n", loop_count);
}
if(loop_count > 9995) {
printf("r: %d\n", loop_count);
}
if(volts > 5.0)  { // Voltage limit: 5V
        volts = 4.99999;
}
if(volts < 0.0) {
        volts = 0.0;
}

out_data = comedi_from_phys(volts, out_range_ptr, out_maxdata);
comedi_data_write(it, out_subdev, out_chan, out_range, aref, out_data);
//u_2 = u_1;
//u_1 = u0;
```

```
                if(loop_count == 10000)
                {
                        loop_count = 0;
                }
                else
                {
                        loop_count++;
                }

                rt_task_wait_period();
        }

…       // Here is an identical code block as the one in Appendix A.2

        for(i=0;i<20000;i++)
        {
                fprintf(tdiff, "%f\n", time_diff[i]);
                //fprintf(tdiff, "                %f\n", current_cpu_time[i]);
        }
        fclose(tdiff);

        …       // Here is an identical code block as the one in Appendix A.2

} // End of recv thread

int main(void)
{
…       // Here is an identical code block as the one in Appendix A.2
}
```

A.4 C Code for Interoperability Suite [69]

```
// Interoperability suite for Wheelchair robot
#include <stdio.h>

…       // Here is an identical code block as the one in Appendix A.1

double u0, u1e, u2e, u3e, u4e, u5e, u6e, u7e, u8e;
char y_8[6]="000000";
double y_0, y_1, y_2, y_3, y_4, y_5, y_6;
double y_7=3;
double u_1, u_2;

//rtai declarations
…       // Here is an identical code block as the one in Appendix A.1

void terminate_normally(int signo) {
…       // Here is an identical code block as the one in Appendix A.1
}
int main(int argc, char *argv[])
{
…       // Here is an identical code block as the one in Appendix A.1

        int sd, clilen;
        int cw, cr;
        int cnt=0;
        char recv_msg[6] = "000000"; // client to interoperability suite
        char fwd[10] = "front";
        char back[10] = "back";
        char stop[10] = "stop";
```

```
char left[10] = "left";
char right[10] = "right";
char *server_ip= "192.168.138.46";
FILE *fp = NULL;
fp = fopen("result.txt","w");
if (fp==NULL) {
        printf("could not open file\n");
        exit(0);
}
RTIME start_time = 0;

…   // Here is an identical code block as the one in Appendix A.1

fprintf(stderr, "binding sockets\n");
server_port = 4444;
second_port = 3333;
addrlen = sizeof(server_addr);
clilen = sizeof(my_addr);
memset((void *) &server_addr, (char) 0, addrlen);
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr(server_ip);
server_addr.sin_port = htons(server_port);
memset((void *) &my_addr, (char) 0, clilen);
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
my_addr.sin_port = htons(second_port);
if ( (bind(sd, (struct sockaddr *) &my_addr, sizeof(my_addr)) < 0) ) {
        perror("2bind() failed ");
        fprintf(stderr, "bind() errno = %d\n", errno);
```

```
                exit(4);
        }
        recv_buffer_size = sizeof(struct recv_data);
        if(( recv_buffer = (struct recv_data *)calloc(1, sizeof(struct recv_data)))
==NULL) {
                fprintf(stderr, "cannot allocate memory for buffer!\n");
                exit(4);
        }
        send_buffer_size = sizeof(struct send_data);
        if(( send_buffer = (struct send_data *)calloc(1, sizeof(struct send_data)))
==NULL) {
                fprintf(stderr, "cannot allocate memory for buffer!\n");
                exit(4);
        }
        fprintf(stderr, "%s: starting blocking message read\n", argv[0]);

…       // Here is an identical code block as the one in Appendix A.1

        start_time = rt_get_cpu_time_ns();
        printf("main: start_time = %lld\n", start_time);
        printf("MASTER TASK STARTS THE ONESHOT TIMER\n");
        //rt_set_oneshot_mode();
        actual_period = start_rt_timer(nano2count(25000));
        printf("actual_period = %lld\n", actual_period);
        printf("MASTER TASK MAKES ITSELF PERIODIC \n");
        rt_task_make_periodic(mtsk, rt_get_time()+ nano2count(3000000),
nano2count(3000000));
        while( 1 ) {
```

```
		cr = recvfrom(sd, recv_msg, 10, 0, (struct sockaddr *) &client_addr,
&clilen);
		if( cr <= -1 ) {
			fprintf(stderr, "2recvfrom() errno = %d\n", errno);
			exit(10);
		}
		start_time = rt_get_cpu_time_ns();
		y_1 = 0;
		y_2 = 0;
		y_3 = 0;
		y_4 = 0;
		y_5 = 0;
		y_6 = 0;
		y_7 = 3;
		y_8[0] = recv_msg[0];
		y_8[1] = recv_msg[1];
		y_8[2] = recv_msg[2];
		y_8[3] = recv_msg[3];
		y_8[4] = recv_msg[4];
		y_8[5] = recv_msg[5];
		u_1 = 0;
		u_2 = 0;
		send_buffer->y_0 = y_0;
		send_buffer->y_1 = y_1;
		send_buffer->y_2 = y_2;
		send_buffer->y_3 = y_3;
		send_buffer->y_4 = y_4;
		send_buffer->y_5 = y_5;
		send_buffer->y_6 = y_6;
```

```
send_buffer->y_7 = y_7;
send_buffer->y_8[0] = y_8[0];
send_buffer->y_8[1] = y_8[1];
send_buffer->y_8[2] = y_8[2];
send_buffer->y_8[3] = y_8[3];
send_buffer->y_8[4] = y_8[4];
send_buffer->u_1 = u_1;
send_buffer->u_2 = u_2;
send_buffer->time_stamp = current_time_stamp;
nw=sendto(sockid, (const void *)send_buffer, send_buffer_size, 0,(struct
sockaddr *) &server_addr, addrlen);
if( nw <= -1 ) {
        perror("1sendto failed ");
        fprintf(stderr, "sendto() errno = %d \n", errno);
        exit(12);
}
nr = recvfrom(sockid, (void *)recv_buffer, recv_buffer_size, 0, (struct
    sockaddr *) &server_addr, &addrlen);
if( nr <= -1 ) {
        fprintf(stderr, "1recvfrom() errno = %d\n", errno);
        exit(10);
}
u0 = recv_buffer->u0;
u1e = recv_buffer->u1e;
u2e = recv_buffer->u2e;
u3e = recv_buffer->u3e;
u4e = recv_buffer->u4e;
u5e = recv_buffer->u5e;
u6e = recv_buffer->u6e;
```

```
                u7e = recv_buffer->u7e;
                u8e = recv_buffer->u8e;
                printf("recv: %s   u0: %f" ,y_8, u0);
                if(u0 == 10.0) {
                        cw = sendto(sd, fwd, 6, 0, (struct sockaddr *) &client_addr,
clilen);
                }
                else if(u0 == 7.0) {
                        cw = sendto(sd, right, 6, 0, (struct sockaddr *) &client_addr,
clilen);
                }
                else if(u0 == 2.0) {
                        cw = sendto(sd, left, 6, 0, (struct sockaddr *) &client_addr,
clilen);
                }
                else if(u0 == 0.0) {
                        cw = sendto(sd, back, 6, 0, (struct sockaddr *) &client_addr,
clilen);
                }
                else if(u0 == 5.0) {
                        cw = sendto(sd, stop, 6, 0, (struct sockaddr *) &client_addr,
clilen);
                }
                end_time = rt_get_cpu_time_ns();
                send_buffer->time_stamp = recv_buffer->time_stamp;
                printf("    end_time - start_time = %lld\n", (end_time - start_time));
                cnt = cnt +1;
        } // End of while
        fclose(fp);
```

```
…       // Here is an identical code block as the one in Appendix A.1

        close(sockid);
        close(sd);
        free(send_buffer);
        free(recv_buffer);
        free(recv_msg);
        free(recv_msg);
}
```

A.5 C++ Code for Client (Wheelchair Robot System) [69]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <NIDAQmx.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#define MAXLOOP 100
#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) ) goto
Error; else
void move(uInt8 direction[8]);
void main(void)
{
        WSADATA w;          // Used to open Windows connection
        SOCKET sd;   // The socket descriptor
        int server_length;       // Length of server struct
        struct sockaddr_in server;      // Information about the server
        struct sockaddr_in client;      // Information about the client
```

```
char *server_ip = "165.91.95.119";
unsigned short server_port = 3333;
char recv_data[6]="wheel", send_data[6]="00000";
uInt8 forward[8]={0,1,1,0,0,1,1,0};
uInt8 backward[8]={1,0,1,0,1,0,1,0};
uInt8 left[8]={0,1,1,0,1,0,1,0};
uInt8 right[8]={1,0,1,0,0,1,1,0};
uInt8 stop[8]={0,0,0,0,0,0,0,0};
int LIFR, LIFRS, RIFR, RIFRS, IFR;
int counter=0;
int32 error=0;
TaskHandle     taskHandle=0;
uInt8 data[8];
char errBuff[2048]={'\0'};
int32 read,bytesPerSamp;
/* Open windows connection */
if (WSAStartup(0x0101, &w) != 0) {
        printf("Could not open Windows connection.\n");
        exit(0);
}
/* Open a datagram socket */
sd = socket(AF_INET, SOCK_DGRAM, 0);
if (sd == INVALID_SOCKET) {
        printf("Could not create socket.\n");
        WSACleanup();
        exit(0);
}
/* Clear out server struct */
memset((void *)&server, '\0', sizeof(struct sockaddr_in));
```

```
        /* Set family and port */
        server.sin_family = AF_INET;
        server.sin_port = htons(server_port);
        server.sin_addr.S_un.S_addr = inet_addr(server_ip);
        /* Clear out client struct */
        memset((void *)&client, '\0', sizeof(struct sockaddr_in));
        /* Set family and port */
        client.sin_family = AF_INET;
        client.sin_port = htons(0);
        client.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
        /* Bind local address to socket */
        if (bind(sd, (struct sockaddr *)&client, sizeof(struct sockaddr_in)) == -1)
        {
                printf("Cannot bind address to socket.\n");
                closesocket(sd);
                WSACleanup();
                exit(0);
        }
        printf("Wheelchair is ready.\n");
        printf("Wheelchair is running.\n");

        // DAQmx Configure Code
        DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
        DAQmxErrChk
(DAQmxCreateDIChan(taskHandle,"Dev1/port1/line0:7","",DAQmx_Val_ChanForAllL
ines));
```

```
	// DAQmx Start Code
	DAQmxErrChk (DAQmxStartTask(taskHandle));

	while (counter<MAXLOOP)
	{
		// DAQmx Read Code
		DAQmxErrChk
(DAQmxReadDigitalLines(taskHandle,1,10.0,DAQmx_Val_GroupByChannel,data,8,&r
ead,&bytesPerSamp,NULL));

		LIFR = data[0];
		LIFRS = data[1];
		RIFR = data[2];
		RIFRS = data[3];
		IFR = data[4];
		send_data[0] = (char)(((int)'0')+LIFR);
		send_data[1] = (char)(((int)'0')+LIFRS);
		send_data[2] = (char)(((int)'0')+RIFR);
		send_data[3] = (char)(((int)'0')+RIFRS);
		send_data[4] = (char)(((int)'0')+IFR);

		// Tranmsit data to get time
		server_length = sizeof(struct sockaddr_in);
		if (sendto(sd, (char *)&send_data, (int)strlen(send_data) + 1, 0, (struct
sockaddr *)&server, server_length) == -1) {
			printf("Error transmitting data.\n");
			closesocket(sd);
			WSACleanup();
			exit(0);
```

```
                }

                // Receive time
                if (recvfrom(sd, (char *)&recv_data, (int)sizeof(recv_data), 0, (struct
sockaddr *)&server, &server_length) < 0) {
                        printf("Error receiving data.\n");
                        closesocket(sd);
                        WSACleanup();
                        exit(0);
                }
                if (strcmp(recv_data,"stop")==0)
                        move(stop);
                else if (strcmp(recv_data,"back")==0)
                        move(backward);
                else if (strcmp(recv_data,"left")==0)
                        move(left);
                else if (strcmp(recv_data,"right")==0)
                        move(right);
                else move(forward);
                sleep(500);
                counter++;
                printf("Current loop: %d.  Command from the server: %s \n", counter,
recv_data);
        }
        move(stop);
        printf("Wheelchair stops. \n");
        closesocket(sd);
        WSACleanup();
        printf("To quit and close the console window, press any key! \n");
```

```
	getchar();

Error:
	if( DAQmxFailed(error) )
		DAQmxGetExtendedErrorInfo(errBuff,2048);
	if( taskHandle!=0 ) {
		// DAQmx Stop Code
		DAQmxStopTask(taskHandle);
		DAQmxClearTask(taskHandle);
	}
	if( DAQmxFailed(error) )
		printf("DAQmx Error: %s\n",errBuff);
}

void move(uInt8 direction[8])
{
	double	error=0;
	TaskHandle  taskHandle=0;
	char	errBuff[2048]={'\0'};

	// DAQmx Configure Code
	DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
	DAQmxErrChk
(DAQmxCreateDOChan(taskHandle,"Dev1/port2/line0:7","",DAQmx_Val_ChanForAll
Lines));

	// DAQmx Start Code
	DAQmxErrChk (DAQmxStartTask(taskHandle));
```

```
        // DAQmx Write Code
        DAQmxErrChk
(DAQmxWriteDigitalLines(taskHandle,1,1,10.0,DAQmx_Val_GroupByChannel,directi
on,NULL,NULL));

Error:
        if( DAQmxFailed(error) )
                DAQmxGetExtendedErrorInfo(errBuff,2048);
        if( taskHandle!=0 ) {
                // DAQmx Stop Code
                DAQmxStopTask(taskHandle);
                DAQmxClearTask(taskHandle);
        }
        if( DAQmxFailed(error) )
                printf("DAQmx Error: %s\n",errBuff);
}
```

A.6 C Code for Server (Dynamic Bandwidth Allocation)

```
#include <stdio.h>

…       // Here is an identical code block as the one in Appendix A.1

// variables for DC motor 1
double y_dot_desi = 7.0;
double e0 = 0; //instead of long
double e1 = 0; //instead of long
double u0 = 0; //value0, instead of long
double u1 = 0; //value1, instead of long
float e_dist = 0; // estimated disturbance
```

```
float h; // sampling time to client

…       // Here is an identical code block as the one in Appendix A.1

main(int argc, char *argv[])
{
…       // Here is an identical code block as the one in Appendix A.1

        while( 1 )
        {
…       // Here is an identical code block as the one in Appendix A.1

                e_dist = u_1; // estimated disturbance calculated from client

…       // Here is an identical code block as the one in Appendix A.1

                if(y_7 == 2) { //Client 2 DC motor
                        if(p2<4000) {  // Choose multiplier for dynamic disturbance
                                dist=0.5; }
                        else if(4000<=p2 && p2 <8000) {
                                dist=1; }
                        else if(8000<=p2 && p2 <12000) {
                                dist=0.333; }
                        else if(12000<=p2 && p2 <16000) {
                                dist=1; }
                        Else {
                                dist=0.667; }
                        e1 = y_dot_desi - y0; // Error Calculation
```

```
                    // Optimal bandwidth allocation for single DC motor
                    if (e_dist<0.067){
                            h = 3; }
                    else if (e_dist>=0.067 && e_dist<0.134) {
                            h = 4; }
                    else if (e_dist>=0.134 && e_dist<0.268) {
                            h = 5; }
                    else if (e_dist>=0.268 && e_dist<0.333) {
                            h = 6; }
                    else if (e_dist>=0.333 && e_dist<0.5) {
                            h = 9; }
                    else if (e_dist>=0.5 && e_dist<0.667) {
                            h = 12; }
                    else if (e_dist>=0.667 && e_dist<1.333) {
                            h = 15; }
                    else if (e_dist>=1.333 && e_dist<2) {
                            h = 21; }
                    else {
                            h = 30; }

                    u0 = u1 + (1.8+3*h1)*e1 - (1.8-3*h1)*e0 + noise[p2]*0.267;   //*
universal digital controller (Tustin's method)
                    send_buffer->u0 = u0;

…       // Here is an identical code block as the one in Appendix A.1

                }
…       // Here is an identical code block as the one in Appendix A.1
        } // END of while
```

```
…       // Here is an identical code block as the one in Appendix A.1

}
```

A.7 C Code for Client (Disturbance Estimation Using PIF)

```
#include <stdio.h>

…       // Here is an identical code block as the one in Appendix A.3

double u_2;
float h=6; // sampling time

// Variables for error variance
float e_var_sum_old=0; // old summation of error variance
float e_var=0; // error variance
float e_var_sum=0; // summation of error variance
float avg_var=0; // average of error variance
float dist=0.3;  //estimated disturbance

// Variables for disturbance estimation using PIF
float wsn[10]={0,0.067,0.134,0.268,0.333,0.5,0.667,1,1.333,2}; /* Amount of Sigma of
White noise */
float
st3[10]={0.0578,0.1200,0.3127,1.0683,1.6097,3.4499,5.8278,10.9706,15.6114,22.1013}
; // 3ms sampling time
float
st4[10]={0.0627,0.1156,0.2569,0.8278,1.2363,2.6693,4.6059,9.0520,13.3866,20.2599};
// 4ms sampling time
```

```
float
st5[10]={0.0857,0.1342,0.2447,0.6985,1.0291,2.1841,3.7752,7.7233,11.8726,18.2277};
// 5ms sampling time
float
st6[10]={0.1327,0.1580,0.2450,0.5801,0.8256,1.6975,2.9038,6.1550,9.7442,16.3004}; //
6ms sampling time
float
st9[10]={0.2745,0.2960,0.3603,0.6069,0.7855,1.4110,2.3069,4.7460,7.8418,14.0000}; //
9ms sampling time
float
st12[10]={0.4973,0.5283,0.5605,0.7404,0.8667,1.3395,1.9777,3.8156,6.2928,11.7949};
// 12ms sampling time
float
st15[10]={0.8368,0.8208,0.8688,1.0006,1.1001,1.4581,1.9609,3.4375,5.3552,10.1955};
// 15ms sampling time
float
st21[10]={1.6506,1.6341,1.6979,1.7702,1.8539,2.0986,2.4540,3.4491,4.7911,8.4559}; //
21ms sampling time
float
st30[10]={3.3833,3.2970,3.2870,3.3857,3.5769,3.5941,3.9642,4.6006,5.4629,7.9140}; //
30ms sampling time

RTIME current_time_stamp;

…       // Here is an identical code block as the one in Appendix A.3

void terminate_normally(int signo);
void endme(int sig)
{
```

```
…       // Here is an identical code block as the one in Appendix A.3
}

void *send_thread_fun(void *arg)
{
…       // Here is an identical code block as the one in Appendix A.3

        // to print sampling time
        float print_samp[100000];
        FILE *fp1 = NULL;
        fp1 = fopen("samp_T_result.txt","w");
        if(fp1 == NULL) {
                printf("could not open file");
                exit(0);
        }
        // to print estimated disturbance
        float print_dist[100000];
        FILE *fp2 = NULL;
        fp2 = fopen("estimated_disturbance.txt","w");
        // to print avg error_variance
        float print_avg_evar[100000];
        FILE *fp3 = NULL;
        fp3 = fopen("avg_e_variance.txt","w");

        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
        …       // Here is an identical code block as the one in Appendix A.3
        while(1)
        {
…       // Here is an identical code block as the one in Appendix A.3
```

```
speed = (cnt*0.214)+1.322; //* DI counting w/ j=1000, PCI-6025E
y_0 = speed;
print_data[loop_count] = y_0;
print_samp[loop_count] = h;  // to print sampling time

// error variance
e_var = (7-y_0)*(7-y_0); // error square
e_var_sum = e_var_sum_old + e_var;
e_var_sum_old = e_var_sum;
// estimate disturbace
if(loop_count%1000 == 0 && loop_count > 0) { /* To avoid calculate
when loop_count=0 (initial value) */
        avg_var=e_var_sum/1000;
        e_var=0;
        e_var_sum=0;
        e_var_sum_old=0;
        if(h==3) {
                if(avg_var<=st3[0]){
                        dist=wsn[0]/st3[0]*avg_var;}
                else if(avg_var<=st3[1]){
                        dist=(wsn[1]-wsn[0])/(st3[1]-st3[0])*(avg_var-
st3[0])+wsn[0];}
                else if(avg_var<=st3[2]){
                        dist=(wsn[2]-wsn[1])/(st3[2]-st3[1])*(avg_var-
st3[1])+wsn[1];}
                else if(avg_var<=st3[3]){
                        dist=(wsn[3]-wsn[2])/(st3[3]-st3[2])*(avg_var-
st3[2])+wsn[2];}
                else if(avg_var<=st3[4]){
```

```
                    dist=(wsn[4]-wsn[3])/(st3[4]-st3[3])*(avg_var-
st3[3])+wsn[3];}
                else if(avg_var<=st3[5]){
                    dist=(wsn[5]-wsn[4])/(st3[5]-st3[4])*(avg_var-
st3[4])+wsn[4];}
                else if(avg_var<=st3[6]){
                    dist=(wsn[6]-wsn[5])/(st3[6]-st3[5])*(avg_var-
st3[5])+wsn[5];}
                else if(avg_var<=st3[7]){
                    dist=(wsn[7]-wsn[6])/(st3[7]-st3[6])*(avg_var-
st3[6])+wsn[6];}
                else if(avg_var<=st3[8]){
                    dist=(wsn[8]-wsn[7])/(st3[8]-st3[7])*(avg_var-
st3[7])+wsn[7];}
                else{
                    dist=2;}
            }
            if(h==4) {
                if(avg_var<=st4[0]){
                    dist=wsn[0]/st4[0]*avg_var;}
                else if(avg_var<=st4[1]){
                    dist=(wsn[1]-wsn[0])/(st4[1]-st4[0])*(avg_var-
st4[0])+wsn[0];}
                else if(avg_var<=st4[2]){
                    dist=(wsn[2]-wsn[1])/(st4[2]-st4[1])*(avg_var-
st4[1])+wsn[1];}
                else if(avg_var<=st4[3]){
                    dist=(wsn[3]-wsn[2])/(st4[3]-st4[2])*(avg_var-
st4[2])+wsn[2];}
```

```
                else if(avg_var<=st4[4]){
                        dist=(wsn[4]-wsn[3])/(st4[4]-st4[3])*(avg_var-
st4[3])+wsn[3];}
                else if(avg_var<=st4[5]){
                        dist=(wsn[5]-wsn[4])/(st4[5]-st4[4])*(avg_var-
st4[4])+wsn[4];}
                else if(avg_var<=st4[6]){
                        dist=(wsn[6]-wsn[5])/(st4[6]-st4[5])*(avg_var-
st4[5])+wsn[5];}
                else if(avg_var<=st4[7]){
                        dist=(wsn[7]-wsn[6])/(st4[7]-st4[6])*(avg_var-
st4[6])+wsn[6];}
                else if(avg_var<=st4[8]){
                        dist=(wsn[8]-wsn[7])/(st4[8]-st4[7])*(avg_var-
st4[7])+wsn[7];}
                else{
                        dist=2;}
        }
        if(h==5) {
                if(avg_var<=st5[0]){
                        dist=wsn[0]/st3[0]*avg_var;}
                else if(avg_var<=st5[1]){
                        dist=(wsn[1]-wsn[0])/(st5[1]-st5[0])*(avg_var-
st5[0])+wsn[0];}
                else if(avg_var<=st5[2]){
                        dist=(wsn[2]-wsn[1])/(st5[2]-st5[1])*(avg_var-
st5[1])+wsn[1];}
                else if(avg_var<=st5[3]){
```

```
                    dist=(wsn[3]-wsn[2])/(st5[3]-st5[2])*(avg_var-
st5[2])+wsn[2];}
                else if(avg_var<=st5[4]){
                    dist=(wsn[4]-wsn[3])/(st5[4]-st5[3])*(avg_var-
st5[3])+wsn[3];}
                else if(avg_var<=st5[5]){
                    dist=(wsn[5]-wsn[4])/(st5[5]-st5[4])*(avg_var-
st5[4])+wsn[4];}
                else if(avg_var<=st5[6]){
                    dist=(wsn[6]-wsn[5])/(st5[6]-st5[5])*(avg_var-
st5[5])+wsn[5];}
                else if(avg_var<=st5[7]){
                    dist=(wsn[7]-wsn[6])/(st5[7]-st5[6])*(avg_var-
st5[6])+wsn[6];}
                else if(avg_var<=st5[8]){
                    dist=(wsn[8]-wsn[7])/(st5[8]-st5[7])*(avg_var-
st5[7])+wsn[7];}
                else{
                    dist=2;}
            }
            if(h==6) {
                if(avg_var<=st6[0]){
                    dist=wsn[0]/st6[0]*avg_var;}
                else if(avg_var<=st6[1]){
                    dist=(wsn[1]-wsn[0])/(st6[1]-st6[0])*(avg_var-
st6[0])+wsn[0];}
                else if(avg_var<=st6[2]){
                    dist=(wsn[2]-wsn[1])/(st6[2]-st6[1])*(avg_var-
st6[1])+wsn[1];}
```

```
                else if(avg_var<=st6[3]){
                        dist=(wsn[3]-wsn[2])/(st6[3]-st6[2])*(avg_var-
st6[2])+wsn[2];}
                else if(avg_var<=st6[4]){
                        dist=(wsn[4]-wsn[3])/(st6[4]-st6[3])*(avg_var-
st6[3])+wsn[3];}
                else if(avg_var<=st6[5]){
                        dist=(wsn[5]-wsn[4])/(st6[5]-st6[4])*(avg_var-
st6[4])+wsn[4];}
                else if(avg_var<=st6[6]){
                        dist=(wsn[6]-wsn[5])/(st6[6]-st6[5])*(avg_var-
st6[5])+wsn[5];}
                else if(avg_var<=st6[7]){
                        dist=(wsn[7]-wsn[6])/(st6[7]-st6[6])*(avg_var-
st6[6])+wsn[6];}
                else if(avg_var<=st6[8]){
                        dist=(wsn[8]-wsn[7])/(st6[8]-st6[7])*(avg_var-
st6[7])+wsn[7];}
                else{
                        dist=2;}
        }
        if(h==9) {
                if(avg_var<=st9[0]){
                        dist=wsn[0]/st9[0]*avg_var;}
                else if(avg_var<=st9[1]){
                        dist=(wsn[1]-wsn[0])/(st9[1]-st9[0])*(avg_var-
st9[0])+wsn[0];}
                else if(avg_var<=st9[2]){
```

```
                    dist=(wsn[2]-wsn[1])/(st9[2]-st9[1])*(avg_var-
st9[1])+wsn[1];}
                else if(avg_var<=st9[3]){
                    dist=(wsn[3]-wsn[2])/(st9[3]-st9[2])*(avg_var-
st9[2])+wsn[2];}
                else if(avg_var<=st9[4]){
                    dist=(wsn[4]-wsn[3])/(st9[4]-st9[3])*(avg_var-
st9[3])+wsn[3];}
                else if(avg_var<=st9[5]){
                    dist=(wsn[5]-wsn[4])/(st9[5]-st9[4])*(avg_var-
st9[4])+wsn[4];}
                else if(avg_var<=st9[6]){
                    dist=(wsn[6]-wsn[5])/(st9[6]-st9[5])*(avg_var-
st9[5])+wsn[5];}
                else if(avg_var<=st9[7]){
                    dist=(wsn[7]-wsn[6])/(st9[7]-st9[6])*(avg_var-
st9[6])+wsn[6];}
                else if(avg_var<=st9[8]){
                    dist=(wsn[8]-wsn[7])/(st9[8]-st9[7])*(avg_var-
st9[7])+wsn[7];}
                else{
                    dist=2;}
            }
            if(h==12) {
                if(avg_var<=st12[0]){
                    dist=wsn[0]/st12[0]*avg_var;}
                else if(avg_var<=st12[1]){
                    dist=(wsn[1]-wsn[0])/(st12[1]-st12[0])*(avg_var-
st12[0])+wsn[0];}
```

```
                else if(avg_var<=st12[2]){
                        dist=(wsn[2]-wsn[1])/(st12[2]-st12[1])*(avg_var-
st12[1])+wsn[1];}
                else if(avg_var<=st12[3]){
                        dist=(wsn[3]-wsn[2])/(st12[3]-st12[2])*(avg_var-
st12[2])+wsn[2];}
                else if(avg_var<=st12[4]){
                        dist=(wsn[4]-wsn[3])/(st12[4]-st12[3])*(avg_var-
st12[3])+wsn[3];}
                else if(avg_var<=st12[5]){
                        dist=(wsn[5]-wsn[4])/(st12[5]-st12[4])*(avg_var-
st12[4])+wsn[4];}
                else if(avg_var<=st12[6]){
                        dist=(wsn[6]-wsn[5])/(st12[6]-st12[5])*(avg_var-
st12[5])+wsn[5];}
                else if(avg_var<=st12[7]){
                        dist=(wsn[7]-wsn[6])/(st12[7]-st12[6])*(avg_var-
st12[6])+wsn[6];}
                else if(avg_var<=st12[8]){
                        dist=(wsn[8]-wsn[7])/(st12[8]-st12[7])*(avg_var-
st12[7])+wsn[7];}
                else{
                        dist=2;}
        }
        if(h==15) {
                if(avg_var<=st15[0]){
                        dist=wsn[0]/st15[0]*avg_var;}
                else if(avg_var<=st15[1]){
```

```
                    dist=(wsn[1]-wsn[0])/(st15[1]-st15[0])*(avg_var-
st15[0])+wsn[0];}
                else if(avg_var<=st15[2]){
                    dist=(wsn[2]-wsn[1])/(st15[2]-st15[1])*(avg_var-
st15[1])+wsn[1];}
                else if(avg_var<=st15[3]){
                    dist=(wsn[3]-wsn[2])/(st15[3]-st15[2])*(avg_var-
st15[2])+wsn[2];}
                else if(avg_var<=st15[4]){
                    dist=(wsn[4]-wsn[3])/(st15[4]-st15[3])*(avg_var-
st15[3])+wsn[3];}
                else if(avg_var<=st15[5]){
                    dist=(wsn[5]-wsn[4])/(st15[5]-st15[4])*(avg_var-
st15[4])+wsn[4];}
                else if(avg_var<=st15[6]){
                    dist=(wsn[6]-wsn[5])/(st15[6]-st15[5])*(avg_var-
st15[5])+wsn[5];}
                else if(avg_var<=st15[7]){
                    dist=(wsn[7]-wsn[6])/(st15[7]-st15[6])*(avg_var-
st15[6])+wsn[6];}
                else if(avg_var<=st15[8]){
                    dist=(wsn[8]-wsn[7])/(st15[8]-st15[7])*(avg_var-
st15[7])+wsn[7];}
                else{
                    dist=2;}
            }

            if(h==21) {
                if(avg_var<=st21[0]){
```

```
                dist=wsn[0]/st21[0]*avg_var;}
            else if(avg_var<=st21[1]){
                dist=(wsn[1]-wsn[0])/(st21[1]-st21[0])*(avg_var-
st21[0])+wsn[0];}
            else if(avg_var<=st21[2]){
                dist=(wsn[2]-wsn[1])/(st21[2]-st21[1])*(avg_var-
st21[1])+wsn[1];}
            else if(avg_var<=st21[3]){
                dist=(wsn[3]-wsn[2])/(st21[3]-st21[2])*(avg_var-
st21[2])+wsn[2];}
            else if(avg_var<=st21[4]){
                dist=(wsn[4]-wsn[3])/(st21[4]-st21[3])*(avg_var-
st21[3])+wsn[3];}
            else if(avg_var<=st21[5]){
                dist=(wsn[5]-wsn[4])/(st21[5]-st21[4])*(avg_var-
st21[4])+wsn[4];}
            else if(avg_var<=st21[6]){
                dist=(wsn[6]-wsn[5])/(st21[6]-st21[5])*(avg_var-
st21[5])+wsn[5];}
            else if(avg_var<=st21[7]){
                dist=(wsn[7]-wsn[6])/(st21[7]-st21[6])*(avg_var-
st21[6])+wsn[6];}
            else if(avg_var<=st21[8]){
                dist=(wsn[8]-wsn[7])/(st21[8]-st21[7])*(avg_var-
st21[7])+wsn[7];}
            else{
                dist=2;}
        }
        if(h==30) {
```

```
                    if(avg_var<=st30[0]){
                        dist=wsn[0]/st30[0]*avg_var;}
                    else if(avg_var<=st30[1]){
                        dist=(wsn[1]-wsn[0])/(st30[1]-st30[0])*(avg_var-
st30[0])+wsn[0];}
                    else if(avg_var<=st30[2]){
                        dist=(wsn[2]-wsn[1])/(st30[2]-st30[1])*(avg_var-
st30[1])+wsn[1];}
                    else if(avg_var<=st30[3]){
                        dist=(wsn[3]-wsn[2])/(st30[3]-st30[2])*(avg_var-
st30[2])+wsn[2];}
                    else if(avg_var<=st30[4]){
                        dist=(wsn[4]-wsn[3])/(st30[4]-st30[3])*(avg_var-
st30[3])+wsn[3];}
                    else if(avg_var<=st30[5]){
                        dist=(wsn[5]-wsn[4])/(st30[5]-st30[4])*(avg_var-
st30[4])+wsn[4];}
                    else if(avg_var<=st30[6]){
                        dist=(wsn[6]-wsn[5])/(st30[6]-st30[5])*(avg_var-
st30[5])+wsn[5];}
                    else if(avg_var<=st30[7]){
                        dist=(wsn[7]-wsn[6])/(st30[7]-st30[6])*(avg_var-
st30[6])+wsn[6];}
                    else if(avg_var<=st30[8]){
                        dist=(wsn[8]-wsn[7])/(st30[8]-st30[7])*(avg_var-
st30[7])+wsn[7];}
                    else{
                        dist=2;}
                }
```

```
                }

                print_dist[loop_count] = dist; // to print estimated disturbance
                print_avg_evar[loop_count] = avg_var; // to print avg error_variance

                send_msg->y_0 = y_0;
                send_msg->y_1 = y_1;
                send_msg->y_2 = y_2;
                send_msg->y_3 = y_3;
                send_msg->y_4 = y_4;
                send_msg->y_5 = y_5;
                send_msg->y_6 = y_6;
                send_msg->y_7 = y_7;
                send_msg->u_1 = dist; // using u_1 to send estimated disturbance
information
                send_msg->u_2 = u_2;
                send_msg->time_stamp = current_time_stamp;

…       // Here is an identical code block as the one in Appendix A.3
        }//END of FOR (WHILE) LOOP

        end_time = rt_get_cpu_time_ns();
…       // Here is an identical code block as the one in Appendix A.3

        for(i=0;i<10000;i++)
        {
                fprintf(fp, "%f\n", print_data[i]);
                printf("%f\n", print_data[i]);
                fprintf(fp1, "%f\n", print_samp[i]); // sampling time
```

```
                fprintf(fp2, "%f\n", print_dist[i]); // estimated disturbance
                fprintf(fp3, "%f\n", print_avg_evar[i]); // average error variance
        }
        fclose(fp);     // result
        fclose(fp1);    // sampling time
        fclose(fp2);    // estimated disturbance
        fclose(fp3);    // avg error variance

…       // Here is an identical code block as the one in Appendix A.3
}       // End of send_thread

void *recv_thread_fun(void *arg)
{
…       // Here is an identical code block as the one in Appendix A.3

        rt_task_make_periodic(mytask, (t0 + nano2count(500500000)),
nano2count(h*1000000)); // h ms sampling time

…       // Here is an identical code block as the one in Appendix A.3

                if(loop_count < 10000)
                {
                u0 = recv_msg->u0;
…       // Here is an identical code block as the one in Appendix A.3
                u8e = recv_msg->u8e;

                h=u1e; // assign sampling period from server

…       // Here is an identical code block as the one in Appendix A.3
```

```
                    }
                    if(loop_count < 9998) {
…       // Here is an identical code block as the one in Appendix A.3
}

int main(void)
{
…       // Here is an identical code block as the one in Appendix A.3
}
```

A.8 C Code for Client (Disturbance Estimation Using ANN)

```
#include <stdio.h>

…       // Here is an identical code block as the one in Appendix A.3

// Variables for error variance
float e_var_old=0; // error variance old
float e_var=0; // error variance
float avg_var=0; // average variance

// Variables for average speed of DC motor
float ysum=0;
float ysum_old=0;
float avg_y=0;

// Variables to calculate ANN
// Input 1
float x1_xoffset[2] = {2.5, 0.1114};
float x1_gain[2] = {0.0727272727272727, 0.161136982548865};
```

```
float x1_ymin = -1;
float h2=0;
float avg_y2=0;
float avg_var2=0;
int k;
// Layer 1
float net1[4]={0,0,0,0};
float net11[4]={0,0,0,0};
float b11[4] = {0.47300776220375267966,1.9122015819874829123,
2.7435818236477609311,-3.9405675003955682456};
float W11[2] = {-0.7688159172855809631,-0.69052644842593469132};
float W12[3] = {-3.7709279503866457439,-0.24358152435790655921};
float W13[3] = {-2.3373025435141800976,0.47755270268816996104};
float W14[3] = {1.4856282135442397951 -4.7080849515018128315};
float a11[4]={0,0,0,0};
// Layer 2
float b21 = -0.71374227756580377324;
float W21[4] = {-7.7056536845927992019,3.2598735695268215018,
2.3522230812494124841,-2.0592920285453231166};
float a21=0;
// Output 1
float y_ymin = -1;
float y_gain = 0.714285714285714;
float y_xoffset = 0;

float dist=0;    //estimated disturbance

RTIME current_time_stamp;
```

```
…       // Here is an identical code block as the one in Appendix A.3

void *send_thread_fun(void *arg)
{
…       // Here is an identical code block as the one in Appendix A.7
// DC motor speed measure
        while(1)
        {
…       // Here is an identical code block as the one in Appendix A.3

                speed = (cnt*0.214)+1.322; //* DI counting w/ j=1000, PCI-6025E
                y_0 = speed;
                print_data[loop_count] = y_0;
                print_samp[loop_count] = h;  // to print sampling time

                // error variance
                e_var = (7-y_0)*(7-y_0);
                e_var = e_var_old + e_var;
                e_var_old = e_var;

                //avg speed
                ysum=ysum_old+y_0;
                ysum_old=ysum;

                if(loop_count%100 == 0 && loop_count > 0) { /* to avoid calculate
when loop_count=0 (initial value) */

                        avg_var=e_var/100;
                        e_var=0;
```

```
                    e_var_old=0;
                    avg_y=ysum/100;
                    ysum=0;
                    ysum_old=0;

                    // Input 1 (Xp1)
                    h2=(h-x1_xoffset[0])*x1_gain[0]+x1_ymin;
                    avg_y2=(avg_y-x1_xoffset[1])*x1_gain[1]+x1_ymin;
                    avg_var2=(avg_var-x1_xoffset[2])*x1_gain[2]+x1_ymin;
                    // Layer 1 (a1)
                    net1[0]=W11[0]*h2+ W11[1]*avg_var2+b11[0];
                    net1[1]=W12[0]*h2+W12[1] *avg_var2+b11[1];
                    net1[2]=W13[0]*h2+W13[1] *avg_var2+b11[2];
                    net1[3]=W14[0]*h2+W14[1] *avg_var2+b11[3];
                    for ( k=0;k<4;k++ ) {
                        net11[k]=fabs(net1[k]);
                        a11[k]=net1[k]/(1+net11[k]); // f(net)=net/(1+|net|) elliot
sigmoidal activation function
                    }
                    // Layer 2 (a2)
        a21=W21[0]*a11[0]+W21[1]*a11[1]+W21[2]*a11[2]+W21[3]*a11[3]+b21;
                    // Output 1 (Y) : estimated disturbance
                    dist=(a21-y_ymin)*y_gain+y_xoffset;

            }
            print_dist[loop_count] = dist; // to print estimated disturbance

…       // Here is an identical code block as the one in Appendix A.7
```

```
}       // End of send_thread

void *recv_thread_fun(void *arg)
{
…       // Here is an identical code block as the one in Appendix A.7
}

int main(void)
{
…       // Here is an identical code block as the one in Appendix A.3
}
```

A.9 C Code for Server (Q-learning)

```
#include <stdio.h>

…       // Here is an identical code block as the one in Appendix A.1

// Variables for DC motor 1
double y_dot_desi = 7.0; //desire speed of DC motor united in RPS
double e0 = 0; //instead of long
double e1 = 0; //instead of long
double u0 = 0; //value0, instead of long
double u1 = 0; //value1, instead of long
float e_dist = 0; // estimated disturbance
float mse = 0; // mse from client
int ri = 0; // index for random disturbance

// Q-learning for single client DC motor
float h[7]={0.003,0.006,0.009,0.012,0.015,0.021,0.030}; // sampling time
```

```
float r[7]={0,0,0,0,0,0,0}; // reward
float q[7]={0,0,0,0,0,0,0}; // Q-values
int size=7;
float alpha=0.25; // learning rate
float gamma=0; // discount factor
float maxq=0; // maximum Q-value
int idx=0;
float yold1=0;
float yold2=0;
float yold3=0;

float h1=0.008; // sampling time of DC1
float h2=0.008; // sampling time of DC2
float h3=0.008; // sampling time of DC3
float h4=0.008; // sampling time of DC4

…       // Here is an identical code block as the one in Appendix A.1

main(int argc, char *argv[])
{
…       // Here is an identical code block as the one in Appendix A.1

        while( 1 )
        {
…       // Here is an identical code block as the one in Appendix A.1

                u_1 = recv_buffer->u_1;
                u_2 = recv_buffer->u_2;
                mse = u_1; // mse from client
```

```
…       // Here is an identical code block as the one in Appendix A.1

            if(y_7 == 2) { //Client 2 DC motor
                e1 = y_dot_desi - y0; // Error Calculation
                u0 = u1 + (1.8+3*h1)*e1 - (1.8-3*h1)*e0 + noise[ri]*0.5;    /*
universal digital controller (Tustin's method) */

                // Q-learning
                if (p2%200==0&&p2>0) {
                    idx=rand()%7;
                    h1=h[idx];
                    if (y<=yold1 && y<=yold2 && y<=yold3){
                        r[idx]=1;} // reward
                    else {
                        r[idx]=0;} // reward
                    maxq=q[0];
                    for (c=1;c<size;c++){
                        if (q[c]>maxq){
                        maxq=q[c];
                    }
                }
                q[idx]=(1-alpha)*q[idx]+alpha*(r[idx]+gamma*maxq); // New Q
                send_buffer->u0 = u0;

…       // Here is an identical code block as the one in Appendix A.1
            }
…       // Here is an identical code block as the one in Appendix A.1
 } // END of while
```

```
…       // Here is an identical code block as the one in Appendix A.1

}
```

APPENDIX B

EXPERIMENTAL DATA OF ERROR VARIANCE BETWEEN REFERENCE INPUT (7 RPS) AND OUTPUT WHEN WHITE GAUSSIAN DISTURBANCE IS INJECTED TO ONE DC MOTOR SYSTEM

Table B-1. Experiment 1

Sampling T. (ms) / Disturbance (m, σ^2)	2.5 ms	3 ms	4 ms	5 ms	6 ms
$(0,0^2)$	0.075	0.061	0.062	0.085	0.137
$(0,0.067^2)$	0.174	0.118	0.111	0.132	0.157
$(0,0.134^2)$	0.406	0.311	0.254	0.239	0.247
$(0,0.268^2)$	1.285	1.070	0.824	0.694	0.579
$(0,0.333^2)$	1.917	1.598	1.231	1.030	0.824
$(0,0.5^2)$	4.088	3.442	2.650	2.167	1.695
$(0,0.667^2)$	6.803	5.833	4.622	3.774	2.901
$(0,1^2)$	12.383	10.781	8.923	7.726	6.171
$(0,1.333^2)$	16.979	15.391	13.211	11.809	9.739
$(0,2^2)$	23.724	21.401	20.398	17.933	16.214
Sampling T. (ms) / Disturbance (m, σ^2)	9 ms	12 ms	15 ms	21 ms	30 ms
$(0,0^2)$	0.279	0.470	0.860	1.533	3.425
$(0,0.067^2)$	0.290	0.511	0.810	1.542	3.206
$(0,0.134^2)$	0.364	0.558	0.860	1.571	3.238
$(0,0.268^2)$	0.607	0.713	0.970	1.681	3.379
$(0,0.333^2)$	0.779	0.849	1.070	1.742	3.531
$(0,0.5^2)$	1.415	1.329	1.376	2.034	3.560
$(0,0.667^2)$	2.319	1.975	1.891	2.399	4.001
$(0,1^2)$	4.750	3.800	3.386	3.408	4.549
$(0,1.333^2)$	7.901	6.339	5.316	4.751	5.452
$(0,2^2)$	14.015	11.774	10.219	8.404	7.877

Table B-2. Experiment 2

Sampling T. (ms) / Disturbance (m, σ^2)	2.5 ms	3 ms	4 ms	5 ms	6 ms
$(0,0^2)$	0.076	0.055	0.061	0.079	0.128
$(0,0.067^2)$	0.172	0.120	0.118	0.133	0.158
$(0,0.134^2)$	0.411	0.312	0.257	0.249	0.247
$(0,0.268^2)$	1.297	1.063	0.826	0.700	0.579
$(0,0.333^2)$	1.910	1.613	1.239	1.026	0.825
$(0,0.5^2)$	4.040	3.453	2.695	2.206	1.699
$(0,0.667^2)$	6.823	5.822	4.597	3.781	2.891
$(0,1^2)$	12.523	11.062	9.084	7.713	6.143
$(0,1.333^2)$	17.234	15.619	13.416	11.908	9.725
$(0,2^2)$	23.854	22.459	20.215	18.478	16.308
Sampling T. (ms) / Disturbance (m, σ^2)	9 ms	12 ms	15 ms	21 ms	30 ms
$(0,0^2)$	0.276	0.506	0.862	1.709	3.363
$(0,0.067^2)$	0.307	0.555	0.839	1.720	3.320
$(0,0.134^2)$	0.363	0.570	0.903	1.784	3.351
$(0,0.268^2)$	0.608	0.771	1.035	1.902	3.403
$(0,0.333^2)$	0.790	0.880	1.134	1.989	3.628
$(0,0.5^2)$	1.414	1.346	1.526	2.216	3.591
$(0,0.667^2)$	2.304	1.987	2.044	2.561	3.950
$(0,1^2)$	4.743	3.822	3.492	3.512	4.633
$(0,1.333^2)$	7.812	6.278	5.391	4.858	5.562
$(0,2^2)$	14.025	11.844	10.215	8.525	8.059

Table B-3. Experiment 3

Sampling T. (ms) / Disturbance (m, σ^2)	2.5 ms	3 ms	4 ms	5 ms	6 ms
$(0,0^2)$	0.076	0.058	0.065	0.093	0.133
$(0,0.067^2)$	0.176	0.122	0.118	0.138	0.160
$(0,0.134^2)$	0.408	0.315	0.260	0.246	0.242
$(0,0.268^2)$	1.310	1.072	0.834	0.702	0.583
$(0,0.333^2)$	1.910	1.617	1.239	1.032	0.828
$(0,0.5^2)$	4.072	3.455	2.663	2.179	1.699
$(0,0.667^2)$	6.778	5.828	4.600	3.770	2.920
$(0,1^2)$	12.519	11.069	9.149	7.731	6.151
$(0,1.333^2)$	17.434	15.825	13.533	11.901	9.768
$(0,2^2)$	23.975	22.443	20.167	18.273	16.379
Sampling T. (ms) / Disturbance (m, σ^2)	9 ms	12 ms	15 ms	21 ms	30 ms
$(0,0^2)$	0.269	0.516	0.789	1.709	3.361
$(0,0.067^2)$	0.291	0.520	0.813	1.641	3.365
$(0,0.134^2)$	0.354	0.554	0.843	1.739	3.272
$(0,0.268^2)$	0.606	0.737	0.997	1.727	3.375
$(0,0.333^2)$	0.788	0.872	1.096	1.831	3.572
$(0,0.5^2)$	1.404	1.343	1.473	2.046	3.632
$(0,0.667^2)$	2.298	1.971	1.948	2.401	3.941
$(0,1^2)$	4.745	3.826	3.434	3.427	4.620
$(0,1.333^2)$	7.813	6.262	5.359	4.765	5.375
$(0,2^2)$	13.960	11.767	10.152	8.440	7.806